Intel® Cluster Toolkit 1.0 Tutorial

Revision 20050222

February 2005

Intel® Cluster Toolkit 1.0 Tutorial
(Revision 200502022)

Table of Contents

1 Disclaimer and Legal Information 2
2 Introduction 3
3 Acronyms and Definitions 4
4 Conventions 5
5 Intel Software Downloads and Installation for the Intel® Cluster Toolkit 6
5.1.1 The Parallel Install Capability for the Intel® Cluster Toolkit Installer 10
6 The Software Architecture of the Intel® Cluster Toolkit 10
7 Getting Started with Intel® MPI Library 13
7.1 Launching MPD Daemons 14
7.1.1 How to Set Up MPD Daemons 14
7.1.2 The mpdboot Command 15
7.2 Compiling and Linking with Intel® MPI Library 15
7.3 Selecting a Network Fabric or Device 16
7.4 Running an MPI Program Using Intel® MPI Library 16
7.5 Experimenting with Intel® MPI Library 16
7.6 Controlling MPI Process Placement 18
8 Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer 19
8.1 Compiling MPI Programs in Conjunction with the Intel® Trace Collector Header Files 21
8.2 Linking MPI Programs with the Intel® Trace Collector Libraries 22
8.2.1 How to Get Additional Help Regarding the Intel® Trace Collector 29
8.3 Experimenting with the MPI Examples in the Directory ${VT_ROOT}/examples 29
8.4 Experimenting with Intel® Trace Analyzer Examples in the Directory ${PAL_ROOT}/examples 33
8.5 How to Get Additional Help Regarding the Intel® Trace Analyzer 34
9 Getting Started in Using the Intel® Cluster Math Kernel Library (Intel® Cluster MKL) 34
9.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK Examples with the Intel® Trace Collector and Intel® Trace Analyzer 38
10 Experimenting with the Intel® MPI Benchmarks 44
11 Hardware and Recommendations for Installation 54
12 System Administrator Checklist 54
13 User Checklist 54

1 Disclaimer and Legal Information

This installation and tutorial document, called Intel® Cluster Toolkit Tutorial, as well as the software described in it, is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

Intel, the Intel logo, Intel SpeedStep, Intel NetBurst, Intel NetStructure, MMX, i386, i486, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Celeron, Intel Centrino, Intel Xeon, Intel XScale, Itanium, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

***** Other names and brands may be claimed as the property of others. All trademarks and registered trademarks referenced in this Intel® Cluster Toolkit document are the property of their respective holders.

Copyright © Intel Corporation 2004 – 2005.

2 Introduction

At the time of this writing, the Intel® Cluster Toolkit 1.0 release consists of:

Intel® MPI Library
Intel® Cluster Math Kernel Library 7.2 (Intel® Cluster MKL 7.2), which includes ScaLAPACK
Intel® Trace Collector[1]
Intel® Trace Analyzer[2]
Intel® MPI Benchmarks

This Intel® Cluster Toolkit 1.0 Tutorial contains information for installing software packages to support homogeneous cluster computing for Intel® Pentium® 4, Intel® Xeon™, Intel® Itanium® 2, and Intel® EM64T processors running Red Hat Enterprise Linux* 3.0. The tutorial was last checked and validated on February 22, 2005. The emphasis of this tutorial is on the *interoperability* of the software components listed above.

The user of the Intel® Cluster Toolkit will mostly likely need assistance from their system administrator in installing the associated software packages on their cluster system. This assumes that the user's login account does not have administrative privileges.

[1] Intel® Trace Collector was formerly known as Vampirtrace.

[2] Intel® Trace Analyzer was formerly known as Vampir.

3 Acronyms and Definitions

The following are acronyms and definitions of those acronyms that are referenced within the Intel® Cluster Toolkit 1.0 User's Guide.

Acronym	Definition
BLACS	Basic Linear Algebra Communication Subprograms – provides a linear algebra oriented message passing interface for distributed memory computing platforms.
DAPL	Direct Access Program Library - an Application Program Interface (API) for Remote Data Memory Access (RDMA).
Ethernet	Ethernet is the predominant local area networking technology. It is transports data over a variety of electrical or optical media. It transports any of several upper layer protocols via data packet transmissions.
GB	Gigabyte
ICT	Intel® Cluster Toolkit
ITA or ita	Intel® Trace Analyzer
ITC or itc	Intel® Trace Collector
MPD	Multiprocessing daemon protocol – a daemon that runs on each node of a cluster. These MPDs configure the nodes of the cluster into a "virtual machine" that is capable of running MPI programs.
MPI	Message Passing Interface - an industry standard, message-passing protocol that typically uses a two-sided send-receive model to transfer messages between processes.
RAM	Random Access Memory
RDMA	Remote Direct Memory Access - this capability allows processes executing on one node of a cluster to be able to "directly" access (execute reads or writes against) the memory of processes within the same user job executing on a different node of the cluster.
RPM	Red Hat Package Management - a system which eases installation, verification, upgrading, and uninstalling Linux packages.
ScaLAPACK	SCAlable LAPACK - an acronym for Scalable Linear Algebra Package or Scalable LAPACK.
SMP	Shared-memory-processor
STF	Structured Trace Format – a trace file format used by the Intel® Trace Collector for efficiently recording data, and this trace format is used by the Intel® Trace Analyzer for performance analysis.
TCP	Transmission Control Protocol - a session-oriented streaming transport protocol which provides sequencing, error detection and correction, flow control, congestion control and multiplexing.

4 Conventions

In this document, italicized text such as:

<directory-path-to-intel-mpi>

refers to a meta-symbol which is typically user dependent. The user will need to replace such meta-symbols with actual user text. An example of this for the above meta-symbol might be:

```
/opt/intel_mpi_10
```

which would be an actual directory path on the user's computing system for an installation of Intel® MPI Library 1.0. The second form of italicized text will be something like:

hostname:n

Again, the user will be required to substitute in a phrase that is specific to the user's computing system, such as:

```
clusternode1:2
```

The last form of italicized notation:

<# of processes>

which is used in the `mpiexec` command:

`mpiexec -n` *<# of processes>* `./mpi_example`

requires a user to supply an integer value that is greater than zero. A user-provided value for the shell command above might be:

```
mpiexec -n 4 ./mpi_example
```

5 Intel Software Downloads and Installation for the Intel® Cluster Toolkit

The Intel® Cluster Toolkit installation process is comprised of seven basic steps. For Intel® Pentium® 4 and Intel® Xeon™ processors, the Intel® Cluster Toolkit 1.0 consists of:

Software Component	Default Installation Directory
Intel® Cluster MKL 7.2	/opt/intel/mkl72cluster
Intel® MPI Library 1.0	/opt/intel_mpi_10
Intel® MPI Benchmarks 2.3	/opt/IMB_2.3
Intel® Trace Analyzer 4.0.3	/opt/intel/ITA_IA32_LIN_AS21_PRODUCT.4.0.3.1
Intel® Trace Collector 5.0	/opt/intel/ITC_IA32_LIN_IMPI_PRODUCT.5.0.1.0

For Intel® Itanium® 2 processors, the Intel® Cluster Toolkit 1.0 consists of:

Software Component	Default Installation Directory
Intel® Cluster MKL 7.2	/opt/intel/mkl72cluster
Intel® MPI Library 1.0	/opt/intel_mpi_10
Intel® MPI Benchmarks 2.3	/opt/IMB_2.3
Intel® Trace Analyzer 4.0.3	/opt/intel/ITA_IA64_LIN_AS21_PRODUCT.4.0.3.1
Intel® Trace Collector 5.0	/opt/intel/ITC_IA64_LIN_IMPI_PRODUCT.5.0.1.0

For Intel® EM64T processors, the Intel® Cluster Toolkit 1.0 consists of:

Software Component	Default Installation Directory
Intel® Cluster MKL 7.2	/opt/intel/mkl72cluster
Intel® MPI Library 1.0	/opt/intel_mpi_10
Intel® MPI Benchmarks 2.3	/opt/IMB_2.3
Intel® Trace Analyzer 4.0.3	/opt/intel/ITA_EM64T_LIN_AS21_PRODUCT.4.0.3.1
Intel® Trace Collector 5.0	/opt/intel/ITC_EM64T_LIN_IMPI_PRODUCT.5.0.2.1

Note that the Intel® Cluster Toolkit installer will automatically make the appropriate selection of binaries, scripts, and text files, from its installation archive based on the Intel processor architecture of the host system where the installation process is initiated. The user does not have to worry about selecting the correct software component names for the given Intel architecture.

The user of the Intel® Cluster Toolkit will mostly likely need assistance from their system administrator (this assumes that the user's login account does not have administrative privileges) in installing the associated software packages on their cluster system. Using an account with system administrator privileges, please perform the following steps:

1. The 4.2.2 version of RPM on Red Hat Enterprise Linux* 3.0 for Itanium® 2 has a broken relocation feature. This will be a serious problem for users trying to do installs on clusters where there are shared devices. A recommended solution is for the user to upgrade to the latest release of RPM.

2. A `machines.LINUX` file will either need to be created, or an existing `machines.LINUX` file can be used by the Intel® Cluster Toolkit installer to deploy the appropriate software packages from the toolkit amongst the nodes of the cluster. This `machines.LINUX` file contains a list of the computing nodes (i.e. the hostnames) for the cluster. The format is one hostname per line:

   hostname

   The hostname should be the same as the result from the Linux* command "`hostname`". An example content for the file `machines.LINUX`, where a contrived cluster consists of eight nodes might be:

   ```
   clusternode1
   clusternode2
   clusternode3
   clusternode4
   clusternode5
   clusternode6
   clusternode7
   clusternode8
   ```

   The text `clusternode1` and `clusternode2`, for example, represent the names of two of the nodes in a contrived computing cluster. The contents of the `machines.LINUX` file can also be used by users to construct an `mpd.hosts` file for the multiprocessing daemon (MPD) protocol. The MPD protocol is used for running MPI applications that utilize Intel® MPI Library.

3. In preparation for doing the installation, the user may want to create a staging area. On the system where the Intel® Cluster Toolkit software components are to be installed, it is recommended that a staging area be constructed in a directory such as `/tmp`. An example folder path staging area might be:

   `/tmp/ict_staging_area`

   where `ict_staging_area` is an acronym for Intel® Cluster Toolkit staging area.

4. Upon purchase of the Intel® Cluster Toolkit, you will receive a serial number (E.g., C111-12345678) for this product. Your serial number can be found within the email receipt of your product purchase. Go to the Intel Registration Center site and provide the product serial number information. Once admission has been granted into the registration center, a user will be able to access the Intel® Premier web pages for software support.

5. The license for the Intel® Cluster Toolkit license file that is provided to the user should be placed in a location pointed to by the `INTEL_LICENSE_FILE` environment variable. Do not change the file name as the "`.lic`" extension is critical. Common locations for the attached license file are:

   `<`*installation path*`>/licenses`

   For example, on the cluster system where the Intel® Cluster Toolkit Software is to be installed, all licenses for Intel-based software products might be placed in:

   `/opt/intel/licenses`

It is also imperative that the user and/or the system administrator set the environment variable `INTEL_LICENSE_FILE` to the directory path where the Intel software licenses will reside *prior* to doing an installation of the Intel® Cluster Toolkit. For Bourne Shell or Korn Shell the syntax for setting the `INTEL_LICENSE_FILE` environment variable might be:

```
export INTEL_LICENSE_FILE=/opt/intel/licenses
```

For C Shell, the syntax might be:

```
setenv INTEL_LICENSE_FILE /opt/intel/licenses
```

6. Patrons can place the Intel® Cluster Toolkit software package into the staging area folder.

7. The installer package for the Intel® Cluster Toolkit has the following general nomenclature:

   l_ict_<*version*>.<*release*>.tar.gz

   where <*version*> is a string such as:

   `b_1.0`, where `b` is an acronym for beta

   or

   `p_1.0`, where `p` is an acronym for production

   The <*release*> meta-symbol is a string such as `017`. This string indicates the package number.

   The command:

   tar –xvzf l_ict_<*version*>.<*release*>.tar.gz

   will create a subdirectory called `l_ict_`<*version*>.<*release*>. Change to that directory with the shell command:

   cd l_ict_<*version*>.<*release*>

   For example, suppose the installation package is called `l_ict_p_1.0.017.tar.gz`. In the staging area that has been created, type the command:

   ```
   tar –xvzf l_ict_p_1.0.017.tar.gz
   ```

   This will create a subdirectory called `l_ict_p_1.0.017`. Change to that directory with the shell command:

   ```
   cd l_ict_p_1.0.017
   ```

   In that folder make sure that `machines.LINUX` file, as mentioned in item 2 above, is either in this directory or the user should know the directory path to this file.

   After making desired adjustments to these files, if any, type the command:

   ```
   ./install
   ```

and follow the prompts issued by this install script. Note that Intel® Trace Analyzer and Intel® MPI Benchmarks are only installed on the master node.

By default, the global root directory for the installation of the Intel® Cluster Toolkit is:

`/opt/intel/ict/<version#>.<minor-version#>`

where `<version#>` is an integer, and `<minor-version#>` is an integer. An example would be 1.0.

Within the folder path `/opt/intel/ict/<version#>.<minor-version#>` one will find the text files:

`ictvars.csh`

`ictvars.sh`

and

`ictsupport.txt`

If one is using Bourne Shell or Korn Shell for the login session, one should type:

```
. ./ictvars.sh
```

and for a login session that uses C-Shell, one should type:

```
source ./ictvars.csh
```

The file called:

`ictsupport.txt`

contains the Package ID and Package Contents information. Please use the information in `ictsupport.txt` when submitting customer support requests.

For the default installation path, an index file, an FAQ file, and the user's guide are located in the directory path:

`/opt/intel/ict/<version#>.<minor-version#>/documentation`

where as mentioned above, `<version#>` is an integer, and `<minor-version#>` is an integer. A complete default folder path to the documentation directory might be:

```
/opt/intel/ict/1.0/documentation
```

The name of the index file is:

`index.htm`

The index file can be used to navigate to the FAQ, the release notes, the user's guide, and an internet accessible Intel® Cluster Toolkit Tutorial.

The name of the FAQ file is:

```
ICT_FAQ_1.0.htm
```

The name of the user's guide file is:

```
ICT_Users_Guide_1.0.pdf
```

By default, the local version of the release notes is located in the directory path:

```
/opt/intel/ict/<version#>.<minor-version#>/release_notes
```

The name of the release notes file is:

```
ICT_Release_Notes_1.0.htm
```

The user may also view the internet accessible Intel® Cluster Toolkit Tutorial web page, which is this document. Note that this web-page document may contain information that is more current than the `ICT_Users_Guide_1.0.pdf` manual that is installed on the user's computing cluster.

5.1.1 The Parallel Install Capability for the Intel® Cluster Toolkit Installer

Installation of the Intel® Cluster Toolkit assumes that the homogenous computing cluster has `ssh` connectivity. When a cluster system is configured in such a way that the cluster toolkit software is to be installed on every node of the cluster, the install script will prompt the user with a query regarding how many parallel installations should go on simultaneously. The parallel installation methodology was developed to make the installation process manageable on large clusters.

As an example, suppose the software is being installed on a four node cluster. The installer could ask the following:

```
There are 3 cluster nodes left to install.  You may select 1 to 3
parallel installations. ( 1-3 ) [ 3 ]:
```

Entering a value of three will direct 3 parallel installations to occur, and will produce a message that looks something like:

```
Do not interrupt this install. Please examine the log files in
/tmp/cluster_toolkit-*.log
Waiting for cluster node installers
```

At the time of this writing, in a worst case scenario, the maximum number simultaneous parallel installations is 8. This maximum value is applicable to node counts greater than or equal to 8. When the installation completes, the user will see the following message:

```
Completed cluster installation
```

6 The Software Architecture of the Intel® Cluster Toolkit

So as to understand the features of the software architecture for the Intel® Cluster Toolkit, one needs to look briefly at the profile infrastructure that is mandated by the MPI Standard. With regards to MPI profiling interfaces, an implementation of the MPI methods *must*[3]:

[3] For complete details, please click on the Profile Interface hyperlink at the URL: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

1. provide a mechanism through which all of the MPI defined functions may be accessed with a name shift. Thus, all of the MPI functions (which normally start with the prefix "`MPI_`") should also be accessible with the prefix "`PMPI_`".
2. ensure that those MPI functions which are not replaced may still be linked into an executable image without causing name clashes.
3. document the implementation of different language bindings of the MPI interface if they are layered on top of each other, so that the profiler developer knows whether they must implement the profile interface for each binding, or can economize by implementing it only for the lowest level routines.
4. where the implementation of different language bindings is done through a layered approach (e.g. the Fortran binding is a set of "wrapper" functions which call the C implementation), ensure that these wrapper functions are separable from the rest of the library. This is necessary to allow a separate profiling library to be correctly implemented, since (at least with Unix-type linker semantics) the profiling library must contain these wrapper functions if it is to perform as expected. This requirement allows the person who builds the profiling library to extract these functions from the original MPI library and add them into the profiling library without bringing along any other unnecessary code.
5. provide a no-op routine `MPI_PCONTROL` in the MPI library.

The above specifications for a profiling interface within MPI can be illustrated as follows in Figure 1. For Figure 1, the `MPI_Send` definition in the profile library is a wrapper function to the call to `PMPI_Send`. When the user's application calls `MPI_Send`, the definition within the profile library is used. In contrast to `MPI_Send`, when `MPI_Recv` is called within the user's executable, the definition of `MPI_Recv` in the MPI library is invoked.

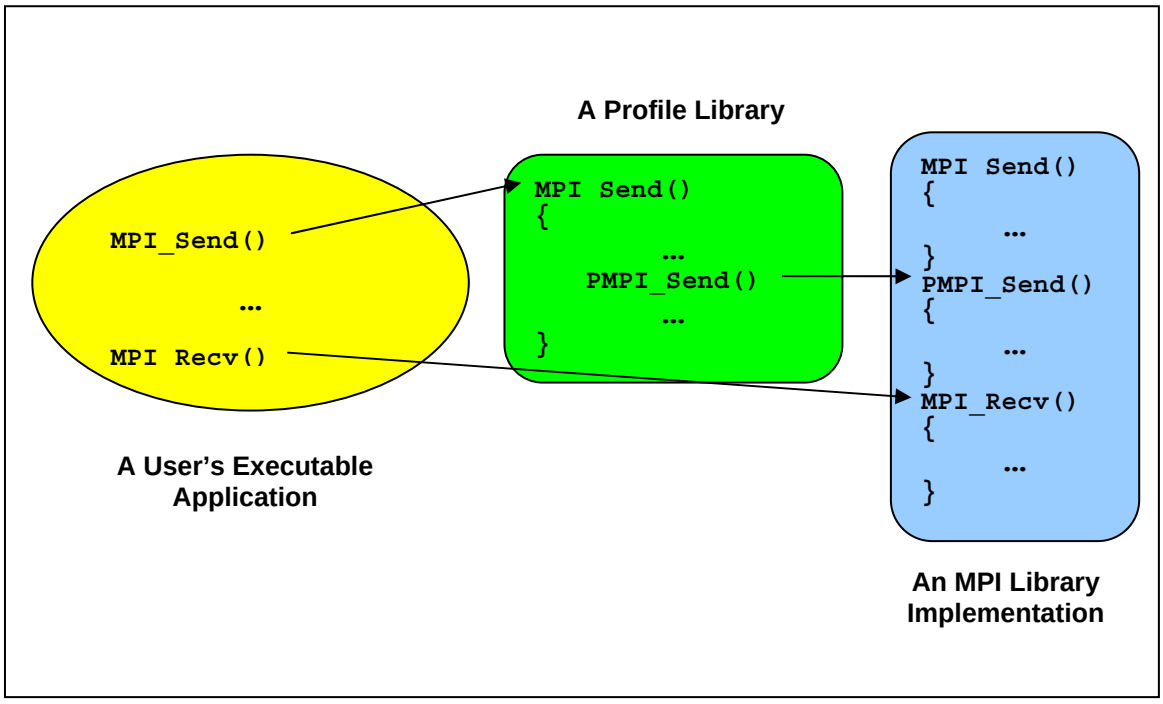

Figure 1 – Implementation of a Profile Library and an MPI Library that Conform to the MPI Standard

An example definition for the profile library function `MPI_Send` might look something like the following:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int
tag, MPI_Comm comm)
{
      int return_value;

      // Start instrumentation

      return_value = PMPI_Send(buf, count, datatype, dest, tag, comm);

      // End instrumentation
}
```

The implementer of the profile library could insert instrumentation before and after the call to `PMPI_Send` so as to gather information about `PMPI_Send`.

Figure 2 takes the abstraction in Figure 1, and illustrates how Intel® MPI Library, the Intel® Trace Collector, and the Intel® Trace Analyzer interact with a user's application. The user can compile their application with Intel® Trace Collector instrumentation. The Intel® Trace Collector Library is analogous to the profile library in Figure 1. They can then proceed to run the executable. Optionally, a debugger can be utilized to resolve logic defects in execution for the application. The instrumentation data related to the Intel® Trace Collector will be placed into a trace file. After the executable completes its tasks, the user can proceed to do post-mortem performance analysis with the Intel® Trace Analyzer.

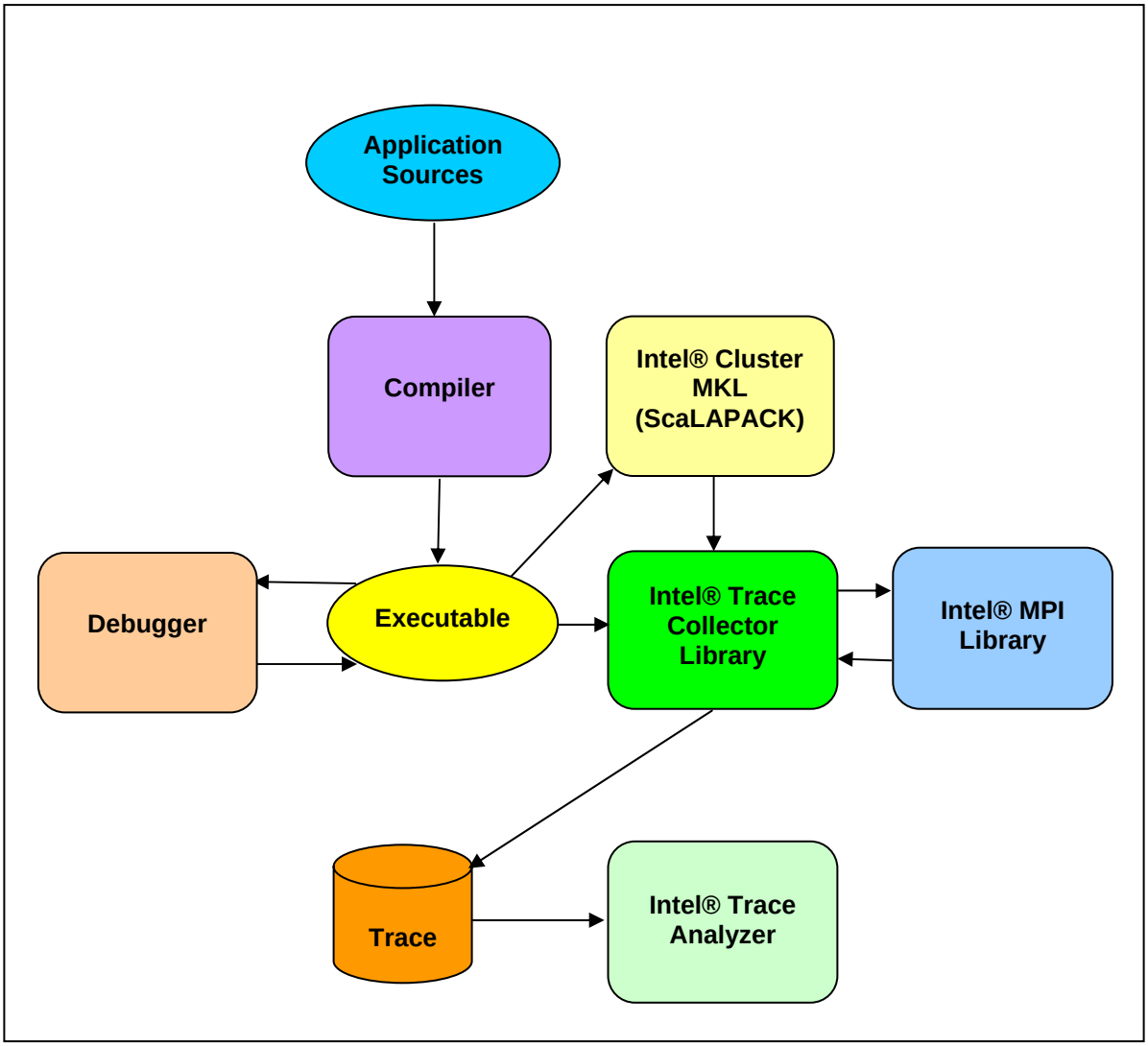

Figure 2 – The Software Architecture of the Intel® Cluster Toolkit

In Figure 2, there can be MPI calls within the executable itself, or the executable can have calls to the ScaLAPACK methods within the Intel® Math Kernel Library. The Basic Linear Algebra Communication Subprograms (BLACS) within ScaLAPACK manages the MPI communication tasks that arise frequently in parallel linear algebra computations.

Following the MPI Standard, the profiling capability of the Intel® Trace Collector Library provides instrumentation wrappers for the MPI methods, which in turn call PMPI methods within the Intel® MPI Library. Thus, one has interoperability capability between Intel® MPI Library, Intel® Cluster Math Kernel Library, Intel® Trace Collector, and Intel® Trace Analyzer.

7 Getting Started with Intel® MPI Library

This chapter will provide some basic information about getting started with Intel® MPI Library. For complete documentation please refer the Intel® MPI Library document `<installdir>/doc/Getting_started.pdf` on the system where Intel® MPI Library is installed.

7.1 Launching MPD Daemons

The Intel® MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. In order to run programs compiled with `mpicc` (or related) commands, you must first set up MPD daemons. It is strongly recommended that users each start and maintain their own set of MPD daemons, as opposed to having the system administrator start up the MPD daemons once for use by all users on the system. This setup enhances system security and gives you greater flexibility in controlling your execution environment.

7.1.1 How to Set Up MPD Daemons

1. Set up environment variables with appropriate values and directories, e.g., in `.cshrc` or `.bashrc` files. At a minimum, set the following environment variables: Ensure that the `PATH` variable includes the following:
   - The `<installdir>/bin` directory. For example, the `<installdir>/bin` directory path should be set.
   - Directory for Python* version 2.2 or greater.
   - If you are using Intel compilers, ensure that the `LD_LIBRARY_PATH` variable contains the directories for the compiler library. You can set this variable by using the `*vars.[c]sh` scripts included with the compiler. Set any additional environment variables your application uses.
2. Create a `$HOME/.mpd.conf` file that contains your MPD password. Your MPD password is not the same as any Linux login password, but rather is used for MPD only. It is an arbitrary password string that is used only to control access to the MPD daemons by various cluster users. To set up your MPD password:

   ```
   password=<your mpd password>
   ```

3. Set protection on the file so that you have read and write privileges, for example, and ensure that the `$HOME/.mpd.conf` file is visible on, or copied to, all the nodes in the cluster as follows:

   ```
   chmod 600 $HOME/.mpd.conf
   ```

4. Verify that `PATH` settings and `mpd.conf` contents can be observed through `rsh` on all nodes in the cluster. For example, use the following commands with each `<node>` in the cluster:

   ```
   rsh <node> env
   rsh <node> cat $HOME/.mpd.conf
   ```

5. Create an `mpd.hosts` text file that lists the nodes in the cluster, with one machine name per line, for use by `mpdboot`.
6. Start up the MPD daemons as follows:

   ```
   mpdboot [ -d -v ] -n <#nodes> [ -f <path/name of mpd.hosts file>]
   ```

   For more information about the `mpdboot` command, see *Mpdboot Command in the section of* `<installdir>/doc/Getting_started.pdf`.
7. Determine the status of the MPD daemons as follows:

   ```
   mpdtrace
   ```

   The output should be a list of nodes that are currently running MPD daemons.

Remarks

- *If required, shut down the MPD daemons as follows:*

  `mpdallexit`.

- *Individual users should start their own set of MPD daemons. It is not recommended to start MPD as root due to setup problems and security issues.*

7.1.2 The mpdboot Command

Use the `mpdboot –f <hosts file>` option to select a specific hosts file to be used. The default is to use `$PWD/mpd.hosts`. A valid host file must be accessible in order for `mpdboot` to succeed.

7.2 Compiling and Linking with Intel® MPI Library

This section describes the basic steps required to compile and link an MPI program, when using the only the *Intel® MPI Library, Development Kit*. To compile and link an MPI program with the Intel® MPI Library:

1. Ensure that the underlying compiler and related software used for `<compiler>` appears in your `PATH`. If you are using Intel® compilers, insure that the compiler library directories appear in `LD_LIBRARY_PATH` environment variable. For example, regarding the Intel® 8.1 compilers, execution of the appropriate set up scripts will do this automatically:

   ```
   /opt/intel_cc_81/bin/iccvars.[c]sh
   ```

   and

   ```
   /opt/intel_fc_81/bin/ifortvars.[c]sh
   ```

2. Compile your MPI program via the appropriate `mpi` compiler command. For example, C code uses the `mpicc` command as follows:

   ```
   mpicc <installdir>/test/test.c
   ```

Other supported compilers have an equivalent command that uses the prefix `mpi` on the standard compiler command. For example, the Intel® MPI Library command for the Intel Fortran compiler (`ifort`) is `mpiifort`.

Remarks

The Command Reference section of `<installdir>/doc/Getting_started.pdf` on the system where Intel® MPI Library is installed *includes additional details on* `mpicc` *and other compiler commands, including commands for other compilers and languages.*

7.3 Selecting a Network Fabric or Device

The Intel® MPI Library supports multiple, dynamically-selectable network fabric device drivers to support different communication channels between MPI processes. The default communication method uses a built-in TCP (Ethernet, or sockets) device driver. Select alternative devices via the command line using the `I_MPI_DEVICE` environment variable. The following network fabric types are supported by Intel® MPI Library:

Possible Interconnection-Device-Fabric Values for the `I_MPI_DEVICE` Environment Variable	**Interconnection Device Fabric Meaning**
`sock`	TCP/Ethernet/sockets (default)
`shm`	Shared-memory only (no sockets)
`ssm`	TCP + shared-memory (for SMP clusters connected via Ethernet)
`rdma[:<provider>]`	InfiniBand*, Myrinet*, etc. (specified *via the* DAPL* provider)
`rdssm[:<provider>]`	TCP + shared-memory + DAPL* (for SMP clusters connected via RDMA-capable fabrics)

7.4 Running an MPI Program Using Intel® MPI Library

Use the `mpiexec` command to launch programs linked with the Intel® MPI Library example:

```
mpiexec -n <# of processes> ./myprog
```

The only required option for the `mpiexec` command is the `-n` option to set the number of processes. If you are a network fabric other than the default fabric (sock), use the `–env` option to specify a value to be assigned to the `I_MPI_DEVICE` variable. For example, to run an MPI program while using the `ssm` device, use the following command:

```
mpiexec -n <# of processes> -env I_MPI_DEVICE ssm ./a.out
```

To run an MPI program while using the `rdma` device, use the following command:

```
mpiexec -n <# of processes> -env I_MPI_DEVICE rdma[:<provider>] ./a.out
```

Any supported device can be selected. See the section titled*, Selecting a Network Fabric or Device in* `<installdir>/doc/Getting_started.pdf`.

7.5 Experimenting with Intel® MPI Library

For the experiments that follow, it is assumed that a computing cluster has at least 2 nodes and there are two SMP processors per node. Start up the MPD daemons by issuing a command such as:

```
mpdboot -n 2 -r rsh -f ~/mpd.hosts
```

Type the command:

```
mpdtrace
```

to verify that there are MPD daemons running on the two nodes of the cluster. The response from issuing this command should be something like:

```
clusternode1
clusternode2
```

assuming that the two nodes of the cluster are called `clusternode1` and `clusternode2`. The actual response will be a function of the user's cluster configuration.

In the *<installdir>*`/test` folder where Intel® MPI Library resides, there are source files for four MPI test-cases. In a local user area, the patron should create a test directory called:

```
test_intel_mpi/
```

From the installation directory of Intel® MPI LIbrary, copy the test files from *<installdir>*`/test` to the directory above. The contents of `test_intel_mpi` should now be:

```
test.c test.cpp test.f test.f90
```

Compile the test applications into executables using the following commands:

```
mpiifort test.f -o testf
mpiifort test.f90 -o testf90
mpiicc test.c -o testc
mpicxx test.cpp -o testcpp
```

Issue the `mpiexec` commands:

```
mpiexec -n 2 ./testf
mpiexec -n 2 ./testf90
mpiexec -n 2 ./testc
mpiexec -n 2 ./testcpp
```

The output from `testcpp` should look something like:

```
Hello world: rank 0 of 1 running on clusternode1
Hello world: rank 1 of 2 running on clusternode2
```

If one has successfully run the above applications using the Intel® MPI Library, the four executables, one can now move the applications from one cluster to another, using different fabrics between the nodes without re-linking. If one encounters problems, please see the section titled *Troubleshooting* within *<installdir>*`/doc/Getting_started.pdf` for possible solutions.

Assuming that the user has an `rdma` device fabric installed on the cluster, the user can issue the following commands for the four executables so as to access that device fabric:

```
mpiexec -n 2 ./testf -env I_MPI_DEVICE rdma
mpiexec -n 2 ./testf90 -env I_MPI_DEVICE rdma
mpiexec -n 2 ./testc -env I_MPI_DEVICE rdma
mpiexec -n 2 ./testcpp -env I_MPI_DEVICE rdma
```

The output from `testf90` using the `rdma` device value for the `I_MPI_DEVICE` environment variable should look something like:

```
Hello world: rank            0  of            2  running on
 clusternode1

Hello world: rank            1  of            2  running on
 clusternode2
```

7.6 Controlling MPI Process Placement

The `mpiexec` command controls how the ranks of the processes are allocated to the nodes in the cluster. By default, `mpiexec` uses round-robin assignment of ranks to the nodes. This placement algorithm may not be the best choice for a user's application, particularly for clusters with SMP nodes.

Suppose that the geometry is <*#ranks*> = 4 and <*#nodes*> = 2, where adjacent pairs of ranks are assigned to each node (for example, for 2-way SMP nodes). Issue the command:

```
cat ~/mpd.hosts
```

The results should be something like:

```
clusternode1
clusternode2
```

Since each node of the cluster is a 2-way SMP, and 4 processes are to be used for the application, the next experiment will distribute the 4 processes such that 2 of the processes will execute on `clusternode1` and 2 will execute on `clusternode2`. For example, one might issue the following commands:

```
mpiexec -n 2 -host clusternode1 ./testf : -n 2 -host clusternode2 ./testf
mpiexec -n 2 -host clusternode1 ./testf90 : -n 2 -host clusternode2 ./testf90
mpiexec -n 2 -host clusternode1 ./testc : -n 2 -host clusternode2 ./testc
mpiexec -n 2 -host clusternode1 ./testcpp : -n 2 -host clusternode2 ./testcpp
```

The following output should be produced for the executable `testc`:

```
Hello world: rank 0 of 4 running on clusternode1
Hello world: rank 1 of 4 running on clusternode1
Hello world: rank 2 of 4 running on clusternode2
Hello world: rank 3 of 4 running on clusternode2
```

In general, if there are *i* nodes in the cluster and each node is *j*-way SMP system, then the `mpiexec` command-line syntax for distributing the *i* by *j* processes amongst the *i* by *j* processors within the cluster is:

```
mpiexec -n j -host <nodename-1> ./mpi_example : \
                -n j -host <nodename-2> ./mpi_example : \
                -n j -host <nodename-3> ./mpi_example : \
                          ...
                -n j -host <nodename-i> ./mpi_example
```

Note that the user would have to fill in appropriate host names for *<nodename-1>* through *<nodename-i>* with respect to their cluster system. For a complete discussion on how to control process placement through the `mpiexec` command, see the *Command Reference* section of *<installdir>*`/doc/Getting_started.pdf`.

To make inquiries about Intel® MPI Library, visit the URL: http://premier.intel.com.

8 Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer

The Intel® Cluster Toolkit installer will place the Intel® Trace Collector package into a directory path on the cluster system such as demonstrated with the following examples:

Intel Processor Architecture	Possible Directory Path for Intel® Trace Collector
Intel® Pentium® 4 and Intel® Xeon™	/opt/intel/ITC-IA32-LIN-IMPI-PRODUCT.5.0.1.0
Intel® Itanium® 2	/opt/intel/ITC_IA64_LIN_IMPI_PRODUCT.5.0.1.0
Intel® EM64T	/opt/intel/ITC_EM64T_LIN_IMPI_PRODUCT.5.0.2.0

In the directory where the Intel® Trace Collector was untarred[7], users should see the following sub-directory and file contents:

```
bin/
ChangeLog
doc/
eula.txt
examples/
include/
install
lib/
man/
README.installation
SilentInstallConfigFile.ini
SilentInstallFParser.awk
slib/
sourceme.csh
sourceme.sh
third_party/
y/
```

Complete user documentation for the Intel® Trace Collector can be found within the file:

<directory-path-to-ITC>/doc/Intel_Trace_Collector_Users_Guide.pdf

on the system where the Intel® Trace Collector is installed.

With respect to the Intel® Trace Analyzer, the Intel® Cluster Toolkit installer should also place the contents of the Intel® Trace Analyzer package into a directory path on the cluster system such as:

Intel Processor Architecture	Possible Directory Path for Intel® Trace Analyzer
Intel® Pentium® 4 and Intel® Xeon™	/opt/intel/ITA_IA32_LIN_AS21_PRODUCT.4.0.3.1
Intel® Itanium® 2	/opt/intel/ITA_IA64_LIN_AS21_PRODUCT.4.0.3.1
Intel® EM64T	/opt/intel/ITA_EM64T_LIN_AS21_PRODUCT.4.0.3.1

In the directory where the Intel® Trace Analyzer was inserted, one should see the following sub-directory and file contents:

```
bin/
ChangeLog
doc/
etc/
examples/
install.sh
man/
SilentInstallFParser.awk
```

The user must add the *<directory-path-to-ITA>*/bin directory to their PATH environment variable. Note that *<directory-path-to-ITA>* is the absolute directory path to either the folder ITA_IA32_LIN_AS21_PRODUCT.4.0.3.1, or the folder ITA_IA64_LIN_AS21_PRODUCT.4.0.3.1, or the folder ITA_EM64T_LIN_AS21_PRODUCT.4.0.3.1. This of course depends on the Intel processor architecture that the user is working with. The version 4.0.3 Intel® Trace Analyzer files that were listed above will reside in one of these folder paths.

Complete user documentation for the Intel® Trace Analyzer can be found within the file:

<directory-path-to-ITA>/doc/Intel_Trace_Analyzer_Users_Guide.pdf

In reference to the Intel® Trace Analyzer and Intel® Trace Collector, the user will need to set environment variables. The environment variable syntax below assumes that the user's login session is using the Bourne Shell or the Korn Shell.

```
export PAL_ROOT=<directory-path-to-ITA>
export LD_ASSUME_KERNEL=2.4.1
. <directory-path-to-ITC>/sourceme.sh
```

For C Shell, the syntax should be:

```
setenv PAL_ROOT <directory-path-to-ITA>
setenv LD_ASSUME_KERNEL 2.4.1
source <directory-path-to-ITC>/sourceme.csh
```

Recall that *<directory-path-to-ITC>* might have an actual setting such as /opt/intel/ict/1.0/ITC_IA64_LIN_IMPI_PRODUCT.5.0.1.0 on the system where the Intel® Trace Collector is installed. Similarly, the environment variable PAL_ROOT points to the root directory path of the Intel® Trace Analyzer installation. The shell script files called sourceme.sh and sourceme.csh that are located in *<directory-path-to-ITC>* on the user's system contain environment variable assignments for VT_ROOT, PATH, and MANPATH.

Note that the above environment variable settings are managed by the shell script files ictvars.sh and ictvars.csh in the directory:

/opt/intel/ict/*<version#>.<release#>*

If one is using Bourne Shell or Korn Shell for the login session, one should type:

```
. ./ictvars.sh
```

and for a login session that uses C-Shell, one should type:

```
source ./ictvars.csh
```

Figure 3 shows a sample of the various panel displays of the Intel® Trace Analyzer that the user can generate when doing post-mortem performance analysis of an application that has been instrumented with the Intel® Trace Collector.

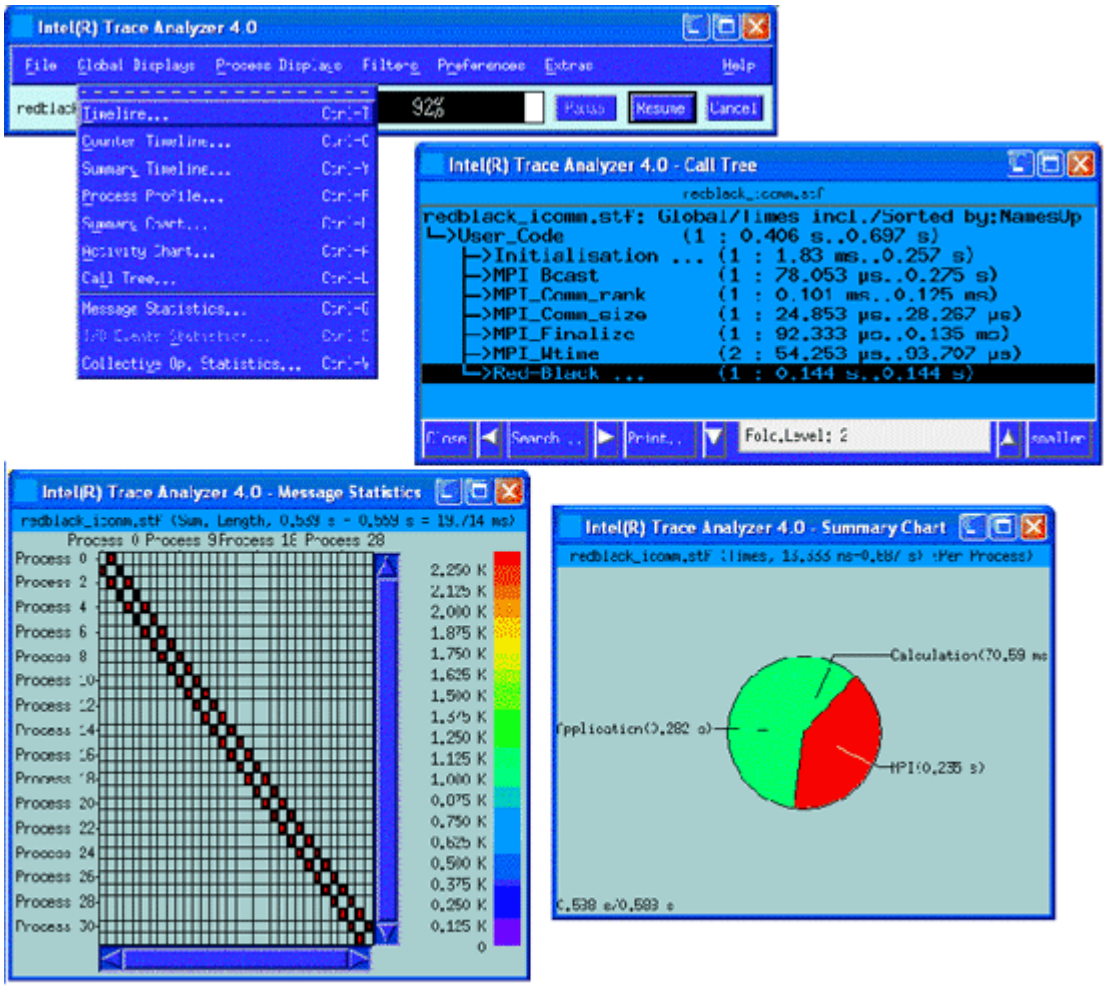

Figure 3 - Multiple Display Panels from the Intel® Trace Analyzer

8.1 Compiling MPI Programs in Conjunction with the Intel® Trace Collector Header Files

User source files without calls to the Intel® Trace Collector APIs can be compiled with the usual methods and without any special precautions. However, source files that do contain calls to the Intel® Trace Collector APIs must include the appropriate header files:

`VT.h` for C and C++
`VT.inc` for Fortran

that exist in the Intel® Trace Collector directory:

<directory-path-to-ITC>`/include`

To compile these source files, the path to the Intel® Trace Collector header files must be passed to the compiler. On most systems, this is done through the compiler command-line with the `-I` flag option. An example of this would be:

```
-I${VT_ROOT}/include
```

8.2 Linking MPI Programs with the Intel® Trace Collector Libraries

The Intel® Trace Collector library, `libVT.a`, contains entry points for all MPI methods. The `libVT.a` library must be linked against the user's application object files before listing Intel® MPI Library or an equivalent. In the section of this user's guide regarding Intel® MPI Library, there was a discussion about creating a user directory called:

```
test_intel_mpi/
```

where the source file contents of `test_intel_mpi` should be:

```
test.c test.cpp test.f test.f90
```

The experimentation which follows will involve linking instrumentation into the MPI applications above. Note, that this instrumentation process is illustrated in Figure 2 with the Intel® Trace Collector and conforms with the model proposed in Figure 1. The following MPI compilation commands should be issued for the MPI source files:

```
mpiifort test.f –g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -
lpthread -lm -o testf_inst
mpiifort test.f90 –g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -
lpthread -lm -o testf90_inst
mpiicc test.c –g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -
lpthread -lm -o testc_inst
mpicxx test.cpp –g -L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -
lpthread -lm -o testcpp_inst
```

Note that with respect to the MPI compilation commands above, the following library options were added to permit instrumentation by an MPI profiling library:

```
-L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lm –lpthread
```

Before running the MPI executable, two Intel® Trace Collector environment variables will be introduced. They are:

```
VT_LOGFILE_PREFIX
```

and

```
VT_PCTRACE
```

The `VT_LOGFILE_PREFIX` environment variable will direct the generation of instrumentation data to an area of the user's choice. The `VT_PCTRACE` environment variable enables runtime program counter tracing. The value of 5 indicates the number of call levels of procedures that will have source location information recorded. Since the unwinding of the call stack each time a function is called can be very costly the setting of the `VT_PCTRACE` environment variable should be handled with discretion. Notice that the `–g` option was added to each of the compilation command-lines listed above. The debug profiling instrumentation that is generated during compilation will be used in conjunction with the `VT_PCTRACE` environment variable to correlate trace events with the source code in the user's application.

Additional documentation about Intel® Trace Collector environment variables can be found in the Intel® Trace Collector folder path:

```
<directory-path-to-ITC>/doc/Intel_Trace_Collector_Users_Guide.pdf
```

For the environment variables mentioned, the following sequence of commands could be issued if the user is running the Bourne Shell:

```
export VT_LOGFILE_PREFIX=${PWD}/inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
export VT_PCTRACE=5
mpiexec -n 2 ./testf_inst
mpiexec -n 2 ./testf90_inst
mpiexec -n 2 ./testc_inst
```

The environment variable `VT_LOGFILE_PREFIX` will be used by the Intel® Trace Collector to place trace data into a folder other than the current directory. For the example above the subfolder is `inst` which is an acronym for instrumentation. The equivalent syntax using C Shell would be:

```
setenv VT_LOGFILE_PREFIX ${PWD}/inst
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
setenv VT_PCTRACE 5
mpiexec -n 2 ./testf_inst
mpiexec -n 2 ./testf90_inst
mpiexec -n 2 ./testc_inst
```

Using the environment variable `VT_LOGFILE_PREFIX` can help prevent working directories from being cluttered with instrumentation data. This is especially true when a user needs to do multiple experiments with Intel® Trace Collector.

The user should list the contents of the `inst` directory after completing the `mpiexec` commands above. The contents of this folder should look something like:

```
.                       testc_inst.stf.sts         testf_inst.prot
..                      testf90_inst.prot          testf_inst.stf
testc_inst.prot         testf90_inst.stf           testf_inst.stf.dcl
testc_inst.stf          testf90_inst.stf.dcl       testf_inst.stf.frm
testc_inst.stf.dcl      testf90_inst.stf.frm       testf_inst.stf.gop
testc_inst.stf.frm      testf90_inst.stf.gop       testf_inst.stf.gop.anc
testc_inst.stf.gop      testf90_inst.stf.gop.anc   testf_inst.stf.msg
testc_inst.stf.gop.anc  testf90_inst.stf.msg       testf_inst.stf.msg.anc
testc_inst.stf.msg      testf90_inst.stf.msg.anc   testf_inst.stf.pr.0
testc_inst.stf.msg.anc  testf90_inst.stf.pr.0      testf_inst.stf.pr.0.anc
testc_inst.stf.pr.0     testf90_inst.stf.pr.0.anc  testf_inst.stf.sts
testc_inst.stf.pr.0.anc testf90_inst.stf.sts
```

Intel® Trace Collector by default partitions the instrumentation data into subcomponents. This model is known as the Structured Trace Format (STF). Hence, the trace file names above have the acronym `stf` embedded within them. The Intel® Trace Collector has the capability to generate these trace file subcomponents in parallel and thus help reduce performance penalties due to instrumentation.

To begin the analysis for the executable `testf90_inst`, type the command:

```
traceanalyzer inst/testf90_inst.stf &
```

This will produce the following two display panels:

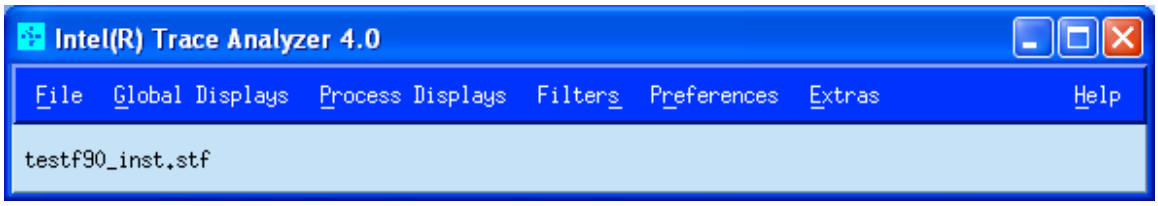

Figure 4 – The Main display panel for Intel® Trace Analyzer

Figure 4 is the Main display panel. It contains menu items for which the user can make selections from. Figure 5 illustrates the Frame Display. This panel displays the Structured Trace File data in a thumbnail type format know as a frame or frames.

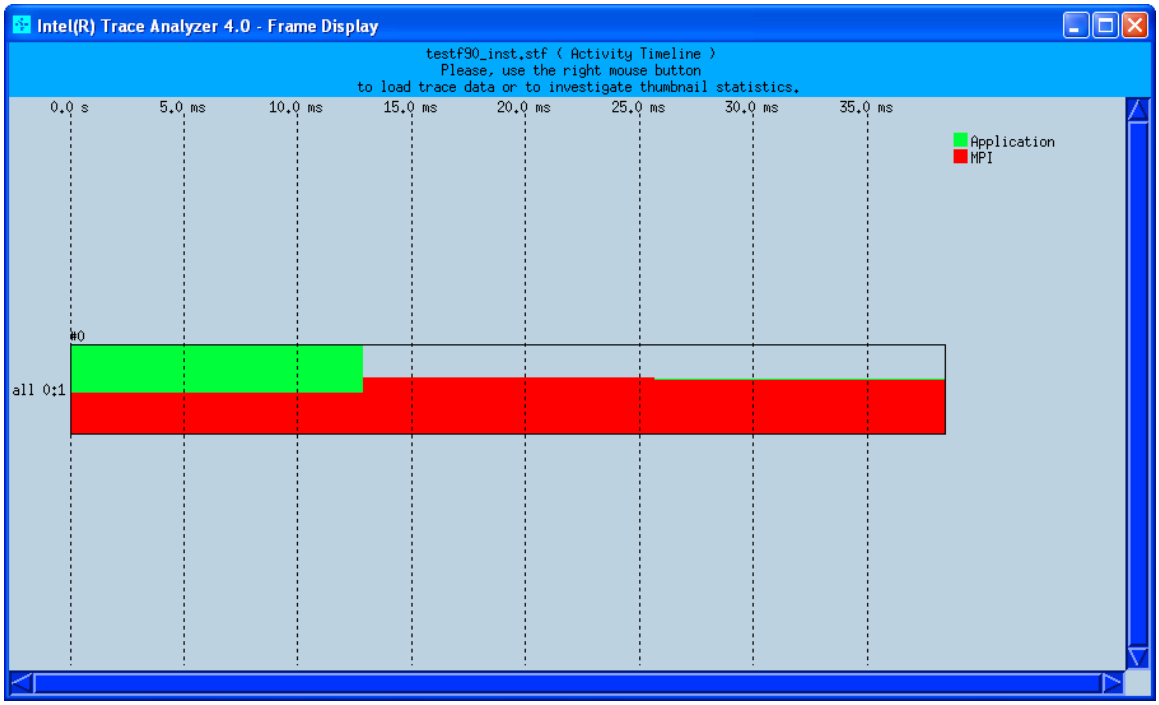

Figure 5 – The Frame Display panel for the Intel® Trace Analyzer

Within the Frame Display, a context menu is available where one should select Load->Whole Trace. This is illustrated in Figure 6.

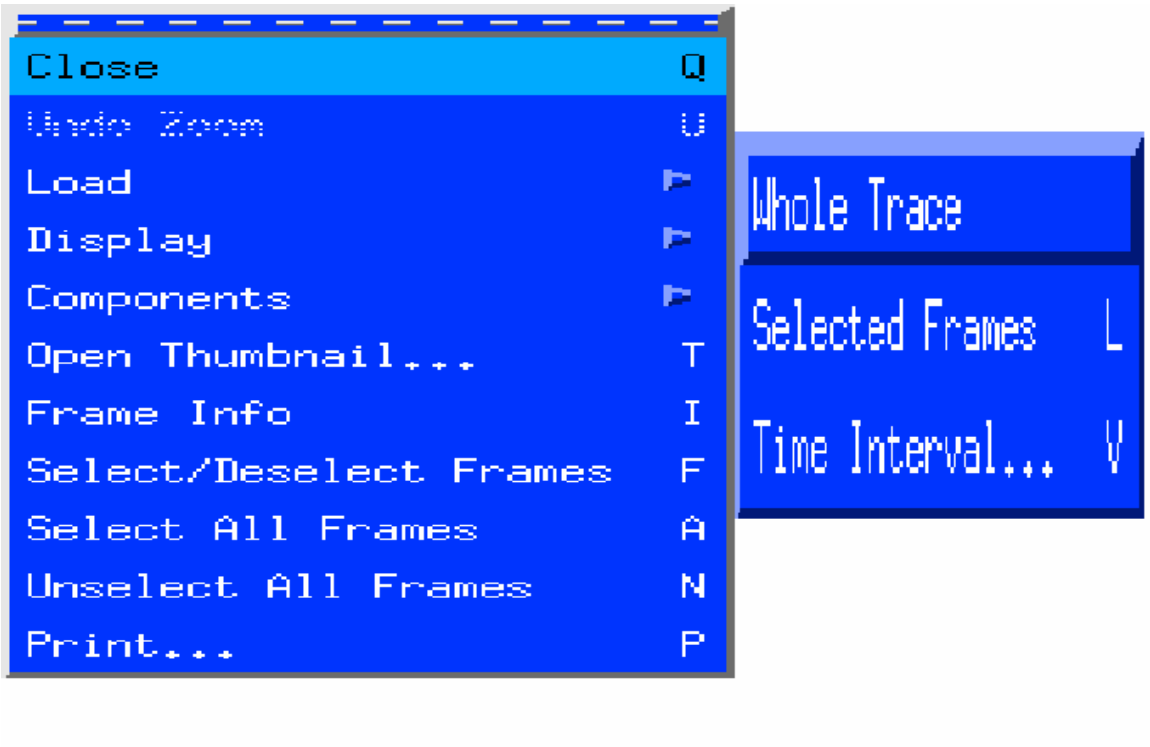

Figure 6 – Context menu for Loading Frames

Upon doing this the whole STF trace is loaded into the Intel® Trace Analyzer. As a result a Summary Chart panel is created (Figure 7). The Summary Chart shows a color-coded histogram. The Sum histogram shows the total computation time of the application. The red histogram is the time spent within the MPI library. The green histogram shows the time that was spent in the user's application. One can add the time contributions of the MPI and User histograms to obtain the Sum bar.

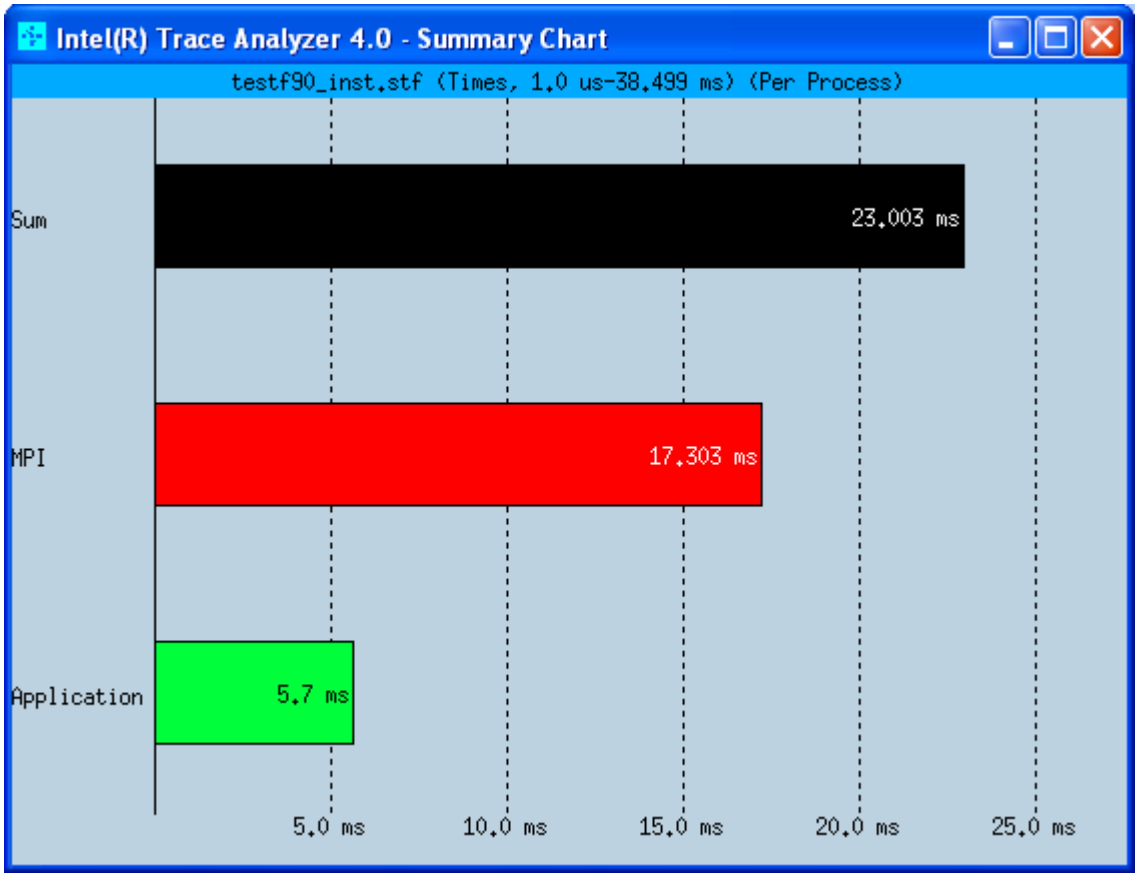

Figure 7– The Summary Chart Display for the executable `testf90_inst`

Upon loading the frames into the Intel® Trace Analyzer, the user can now go back to the Main panel and select Global Displays->Timeline. As a result, the Timeline window is generated as illustrated in Figure 8.

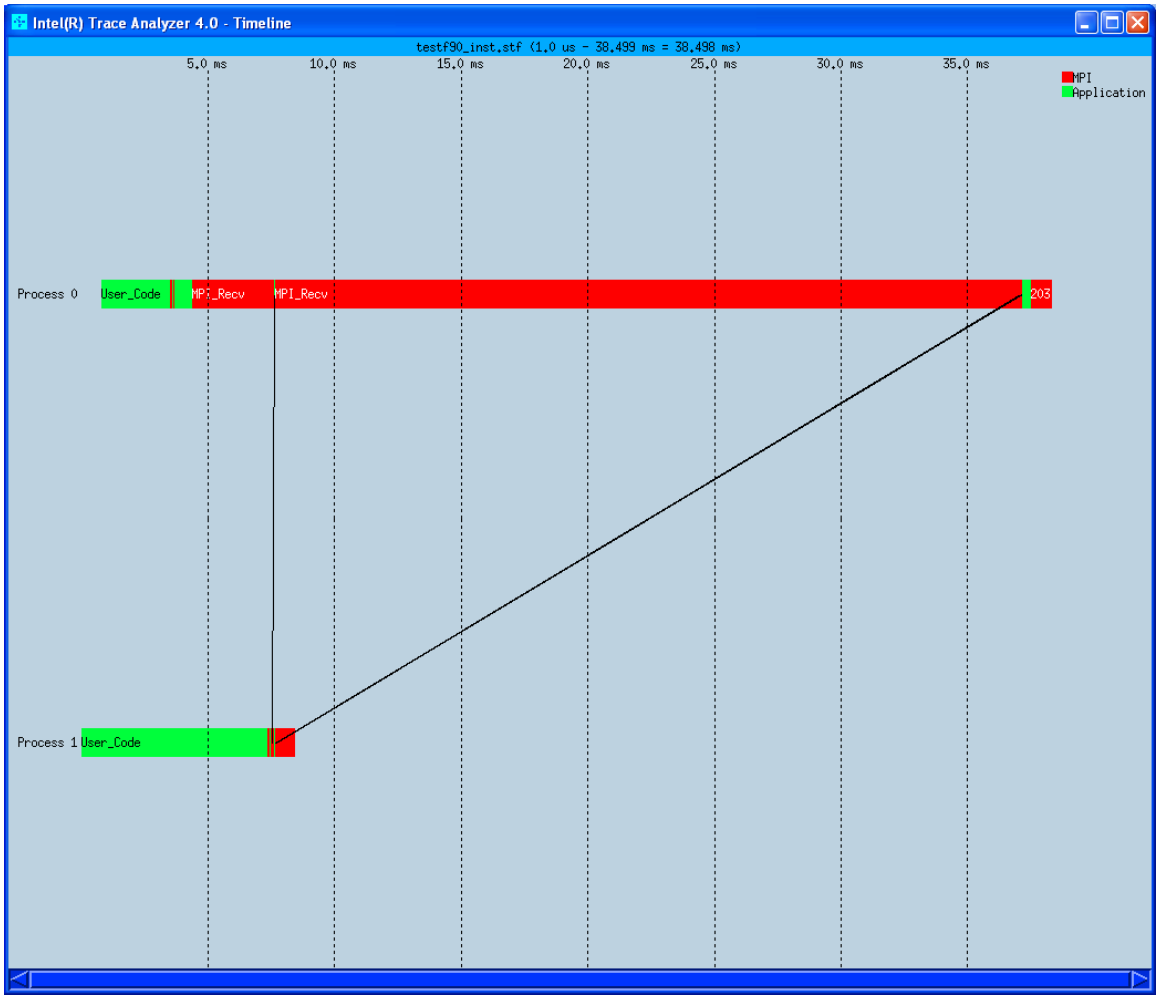

Figure 8 – The Timeline Window for the executable `testf90_inst`

In Figure 8, the black lines signify message events that took place over time for the MPI application. If one moves the mouse so that the cross-hairs line up with the message event that took place at about 7.5 milliseconds, and the leftmost mouse button is depressed, then an Identified Message window will appear as in Figure 9.

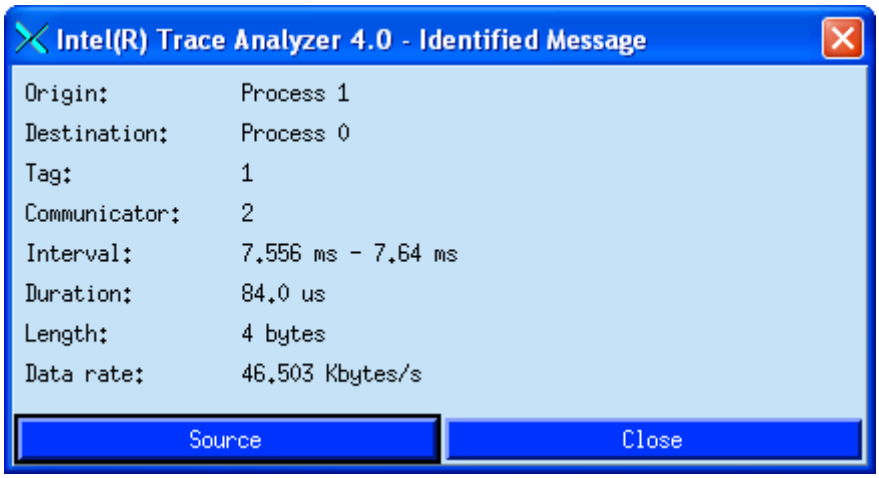

Figure 9 – Identified Message Panel Display

If the user presses the Source button on this panel, then two new display windows will appear which essentially do a drill-down to the application source so as to show what programming semantics triggered the message event.

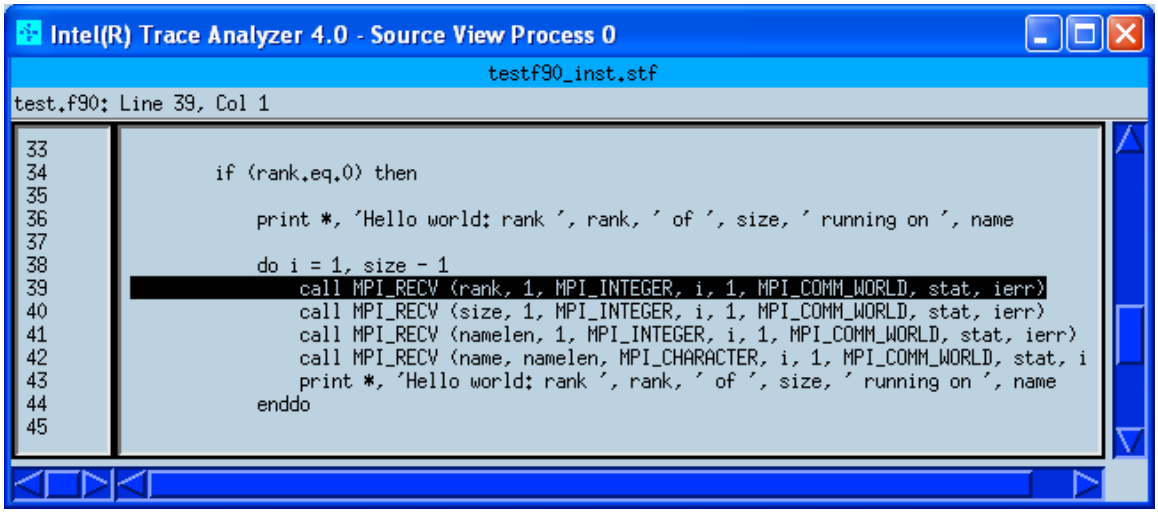

Figure 10 – The Source View for Process 0

Figure 10 illustrates that process 0 was executing an MPI_RECV call, and Figure 11 shows that process 1 was executing an MPI SEND call when this message event took place in Figure 10.

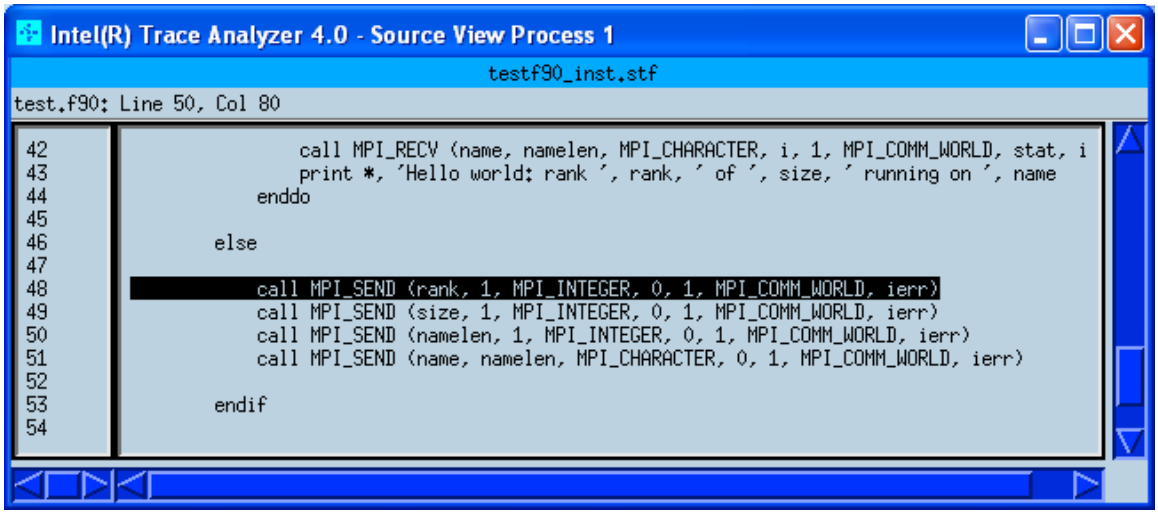

Figure 11 – The Source View for Process 1

The user can terminate the Intel® Trace Analyzer session by going to the Main panel and select the menu path File->Exit.

The user can proceed to issue the shell command:

```
traceanalyzer inst/testc_inst.stf &
```

and repeat the same trace analyzer commands that were used for `inst/testf90_inst.stf`.

8.2.1 How to Get Additional Help Regarding the Intel® Trace Collector

Recall that complete user documentation for the Intel® Trace Collector can be found within the file:

`<directory-path-to-ITC>/doc/Intel_Trace_Collector_Users_Guide.pdf`

Also, note that *`<directory-path-to-ITC>`* is the absolute directory path to the folder such as `ITC_IA64_LIN_IMPI_PRODUCT.5.0.1.0`, which is where the `doc/Intel_Trace_Collector_Users_Guide.pdf` file resides, that was listed above. To make inquiries about the Intel® Trace Collector, visit the URL: http://premier.intel.com.

8.3 Experimenting with the MPI Examples in the Directory `${VT_ROOT}/examples`

Recall that there exists a directory called `${VT_ROOT}/examples`. The user should make a copy of this directory into a local directory partition where experimentation can take place. This local directory partition should be accessible to all nodes in the cluster. Within the sub-directory called examples, there is a script called `palmake`. If the user types the command:

```
./palmake
```

the executables called `vtjacobic`, `vtjacobif`, `vtallpair`, and `vtallpairc` will be created. Additional information about the shell script called `palmake` can be found in the text file called `README.examples` within the examples sub-directory. Assuming that the user has set up an appropriate port number with the `chp4_servs` command, and the user environment variables

`MPI_USEP4SSPORT` and `MPI_P4SSPORT` have been initialized, the user can type an `mpiexec` command such as:

```
mpiexec -n 2 vtallpair
```

After this MPI application completes execution, there will be a collection of analysis files that look something like:

```
vtallpair.prot
vtallpair.stf
vtallpair.stf.dcl
vtallpair.stf.frm
vtallpair.stf.gop
vtallpair.stf.gop.anc
vtallpair.stf.msg
vtallpair.stf.msg.anc
vtallpair.stf.pr.0
vtallpair.stf.pr.0.anc
vtallpair.stf.sts
```

The user can proceed to visualize the analysis data with the command:

```
traceanalyzer vtallpair.stf &
```

As with the previous tracing examples, two Intel® Trace Analyzer panels will immediately appear. The first panel is called the main display (Figure 12), and the second window is called the frame panel (Figure 13).

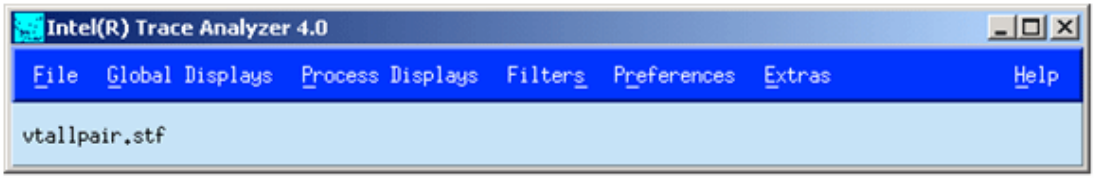

Figure 12 - Intel® Trace Analyzer Main Panel

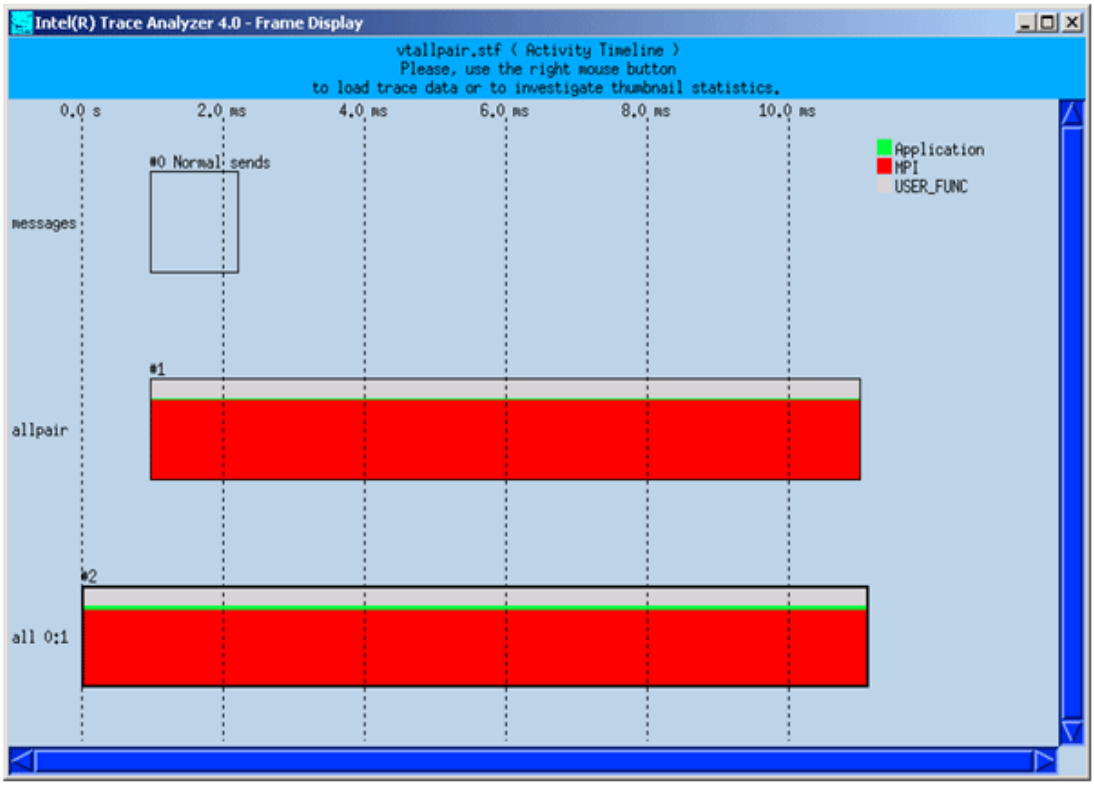

Figure 13 - Intel® Trace Analyzer Frame Display

Within the Frame Display (Figure 13), the user should proceed to press on the right mouse button to force a context menu to appear. For this context menu, the user should follow the selection path Load -> Whole Trace. As a result of this context menu selection, the Summary Chart display panel will appear (Figure 14).

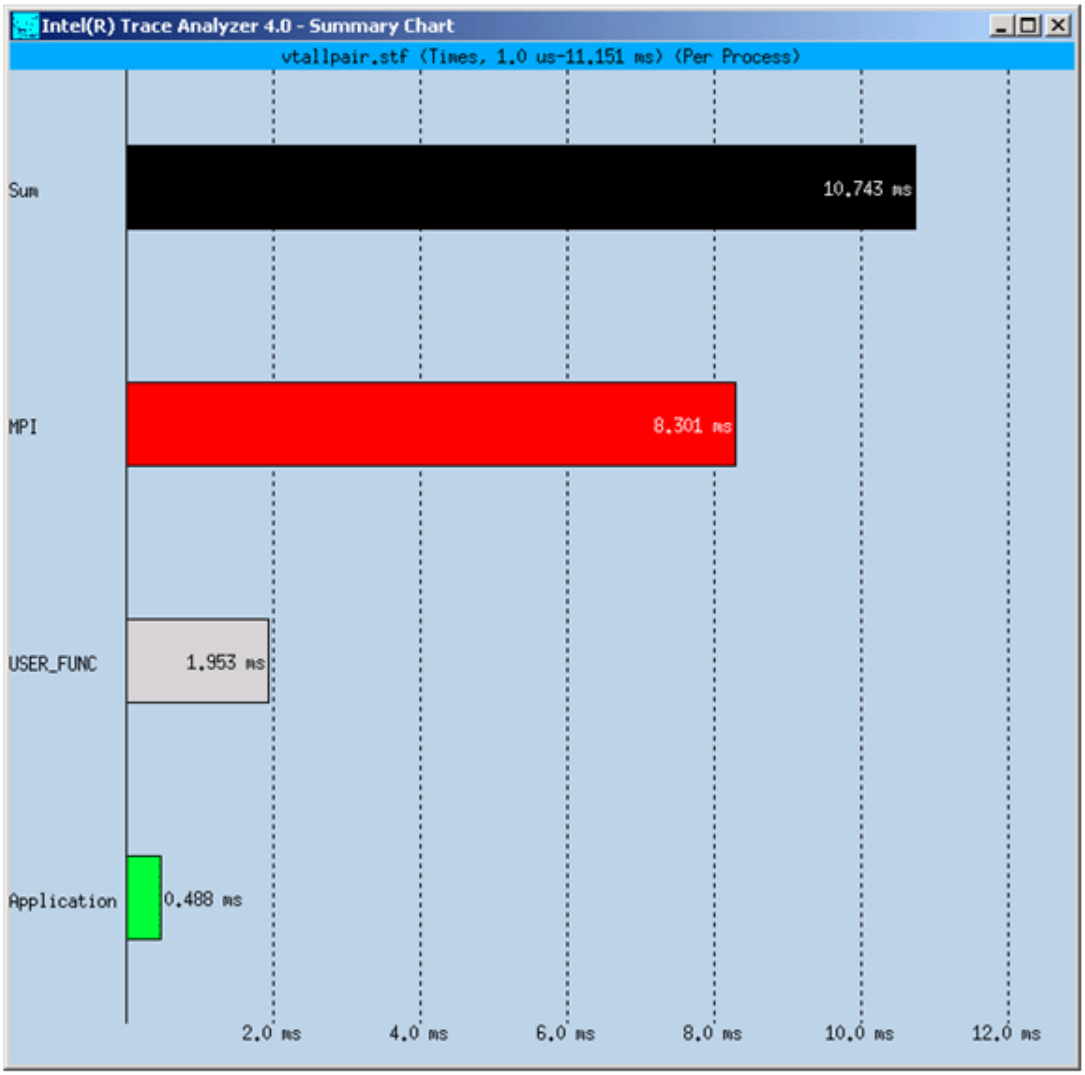

Figure 14 - Intel® Trace Analyzer Summary Chart

If one proceeds to go to the Main panel display (Figure 12), and uses the menu selection Global Displays -> Timeline, then a Timeline panel will appear. This Timeline display looks something like what is illustrated in Figure 15.

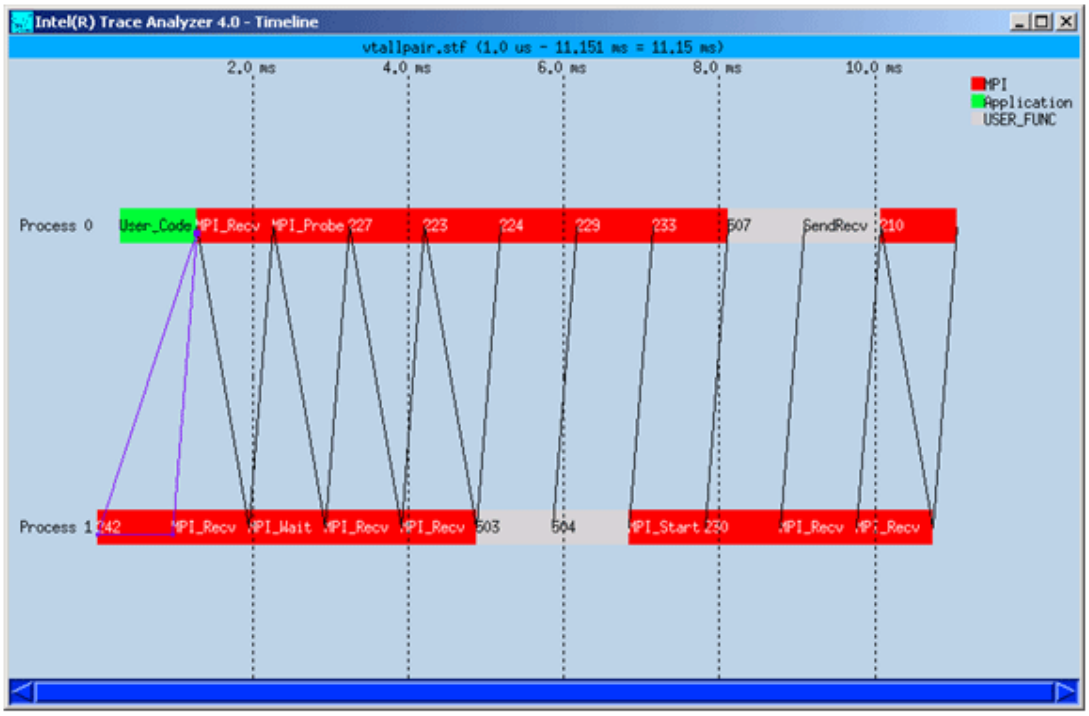

Figure 15 - Intel® Trace Analyzer Timeline Display

8.4 Experimenting with Intel® Trace Analyzer Examples in the Directory `${PAL_ROOT}/examples`

If one goes to the directory `${PAL_ROOT}/examples`, one will find the following key trace analysis files:

```
redblack_icomm.stf
redblack_sndrcv.stf
sort_lin.stf
sort_tree.short.stf
sort_tree.stf
```

To begin the analysis of the trace data one can simply type a command such as:

```
traceanalyzer redblack_sndrcv.stf &
```

If one proceeds to load all of the frames (See Figure 13), and then follow the main panel menu path Global Displays -> Timeline, then the following window will appear (Figure 16).

Figure 16 - Intel® Trace Analyzer Display for 32 MPI Processes

The given application that is being analyzed used 32 MPI processes during execution.

8.5 How to Get Additional Help Regarding the Intel® Trace Analyzer

Recall that complete user documentation for the Intel® Trace Analyzer can be found within the file:

`<directory-path-to-ITA>/doc/Intel_Trace_Analyzer_Users_Guide.pdf`

Also, note that `<directory-path-to-ITA>` is the absolute directory path to a folder such as `ITA_IA64_LIN_AS21_PRODUCT.4.0.3.1`, which is where the `doc/Intel_Trace_Analyzer_Users_Guide.pdf` file resides, that was listed above. To make inquiries about the Intel® Trace Analyzer, visit the URL: http://premier.intel.com.

9 Getting Started in Using the Intel® Cluster Math Kernel Library (Intel® Cluster MKL)

The installation process for Intel® Cluster MKL on the cluster system will produce a sub-directory called `mkl72cluster`. The default directory path for the library installation process is:

`/opt/intel/mkl72cluster`

The sub-contents of the `mkl72cluster` sub-directory should be:

```
doc/
examples/
include/
lib/
mkllic.htm
mklnotes.htm
redist.txt
tests/
tools/
uninstall.sh
```

Complete user documentation for the Intel® Cluster Math Kernel Library 7.2 can be found within the directory path:

<directory-path-to-mkl>/doc

where *<directory-path-to-mkl>* is the absolute directory path to where the Intel® Cluster MKL files and sub-directories are installed on the cluster system.

The user should go the following directory:

<directory-path-to-mkl>/tools/environment

and issue the Linux command:

```
. ./mklvars64.sh
```

if the user is using the Bourne or Korn command-line shell. The sourcing of this shell script will respectively update the user environment variables, `INCLUDE` and `LD_LIBRARY_PATH`.

On Red Hat* Enterprise Linux 3.0, in order to ensure that the correct support libraries are linked, the environment variable `LD_ASSUME_KERNEL` must be set. This environment variable was referenced in the installation section for Intel® Trace Collector and Intel® Trace Analyzer. The syntax for this environment variable might be:

```
export LD_ASSUME_KERNEL=2.4.1
```

In the directory path:

<directory-path-to-mkl>/tests/scalapack

the user can type the command:

```
make lib64 F=intel80 LIBdir=<directory-path-to-mkl>/lib/64
```

Note that the `make` command above is applicable to Itanium® 2-based systems. This makefile creates and runs executables for the ScaLAPACK (SCAlable LAPACK) examples.

`<`*`directory-path-to-mkl`*`>/tests/scalapack/source/TESTING`

The user can invoke an editor to view the results in each of the "`*.txt`" files that have been created. As an example result, the file "`slu_ipf_intel80_noopt.txt`" might have something like the following in terms of contents for a run on a cluster using 4 MPI processes. The cluster that generated this sample output consisted of 2 nodes with 2 processors per node.

```
ScaLAPACK Ax=b by LU factorization.
'MPI Machine'

Tests of the parallel real single precision LU factorization and solve.
The following scaled residual checks will be computed:
 Solve residual= ||Ax - b|| / (||x|| * ||A|| * eps * N)
 Factorization residual = ||A - LU|| / (||A|| * eps * N)
The matrix A is randomly generated for each test.

An explanation of the input/output parameters follows:
TIME    : Indicates whether WALL or CPU time was used.
M       : The number of rows in the matrix A.
N       : The number of columns in the matrix A.
NB      : The size of the square blocks the matrix A is split into.
NRHS    : The total number of RHS to solve for.
NBRHS   : The number of RHS to be put on a column of processes before going
 on to the next column of processes.
P       : The number of process rows.
Q       : The number of process columns.
THRESH  : If a residual value is less than THRESH, CHECK is flagged as PASSED
LU time : Time in seconds to factor the matrix
Sol Time: Time in seconds to solve the system.
MFLOPS  : Rate of execution for factor and solve.

The following parameter values will be used:
  M       :  512  1001  1555  2010
  N       :  489  1221  1528  2002
  NB      :   61   128
  NRHS    :    1     8
  NBRHS   :    1    11
  P       :    1     2     1     4
  Q       :    1     2     4     1

Relative machine precision (eps) is taken to be       0.596046E-07
Routines pass computational tests if scaled residual is less than   1.0000

TIME     M     N  NB NRHS NBRHS    P    Q  LU Time Sol Time  MFLOPS  CHECK
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- ------

WALL   512   489  61     0    0    1    1     0.06     0.00  1453.22 PASSED
WALL   512   489 128     0    0    1    1     0.06     0.00  1453.22 PASSED
WALL  1001  1221  61     0    0    1    1     0.43     0.00  2066.15 PASSED
WALL  1001  1221 128     0    0    1    1     0.42     0.00  2126.91 PASSED
```

```
WALL  1555  1528  61     0    0    1    1     1.12     0.00  2182.28 PASSED
WALL  1555  1528 128     0    0    1    1     1.09     0.00  2239.70 PASSED
WALL  2010  2002  61     0    0    1    1     2.37     0.00  2272.20 PASSED
WALL  2010  2002 128     0    0    1    1     2.29     0.00  2349.43 PASSED
WALL   512   489  61     0    0    2    2     0.18     0.00   473.14 PASSED
WALL   512   489 128     0    0    2    2     0.09     0.00   884.57 PASSED
WALL  1001  1221  61     0    0    2    2     0.29     0.00  3055.57 PASSED
WALL  1001  1221 128     0    0    2    2     0.35     0.00  2522.62 PASSED
WALL  1555  1528  61     0    0    2    2     0.64     0.00  3794.66 PASSED
WALL  1555  1528 128     0    0    2    2     0.75     0.00  3237.83 PASSED
WALL  2010  2002  61     0    0    2    2     1.11     0.00  4864.19 PASSED
WALL  2010  2002 128     0    0    2    2     1.19     0.00  4513.17 PASSED
WALL   512   489  61     0    0    1    4     0.09     0.00   884.57 PASSED
WALL   512   489 128     0    0    1    4     0.07     0.00  1271.57 PASSED
WALL  1001  1221  61     0    0    1    4     0.26     0.00  3389.77 PASSED
WALL  1001  1221 128     0    0    1    4     0.37     0.00  2410.50 PASSED
WALL  1555  1528  61     0    0    1    4     0.58     0.00  4225.26 PASSED
WALL  1555  1528 128     0    0    1    4     0.73     0.00  3346.97 PASSED
WALL  2010  2002  61     0    0    1    4     0.99     0.00  5449.51 PASSED
WALL  2010  2002 128     0    0    1    4     1.30     0.00  4129.98 PASSED
WALL   512   489  61     0    0    4    1     1.38     0.00    60.55 PASSED
WALL   512   489 128     0    0    4    1     0.64     0.00   129.59 PASSED
WALL  1001  1221  61     0    0    4    1     2.67     0.00   332.23 PASSED
WALL  1001  1221 128     0    0    4    1     2.62     0.00   338.98 PASSED
WALL  1555  1528  61     0    0    4    1     4.67     0.00   522.14 PASSED
WALL  1555  1528 128     0    0    4    1     4.74     0.00   514.92 PASSED
WALL  2010  2002  61     0    0    4    1     6.72     0.00   800.81 PASSED
WALL  2010  2002 128     0    0    4    1     6.40     0.00   840.80 PASSED

Finished     32 tests, with the following results:
   32 tests completed and passed residual checks.
    0 tests completed and failed residual checks.
    0 tests skipped because of illegal input values.


END OF TESTS.
```

The text in the table above reflects the actual output that a user will see.

Please note that the above results are dependent on factors such as the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster. Therefore, the results will vary from cluster configuration to cluster configuration.

If one proceeds to load the table above into a Microsoft* Excel Spreadsheet, and builds a chart to compare the LU Time and the Megaflop values, one might see something like the following (Figure 17):

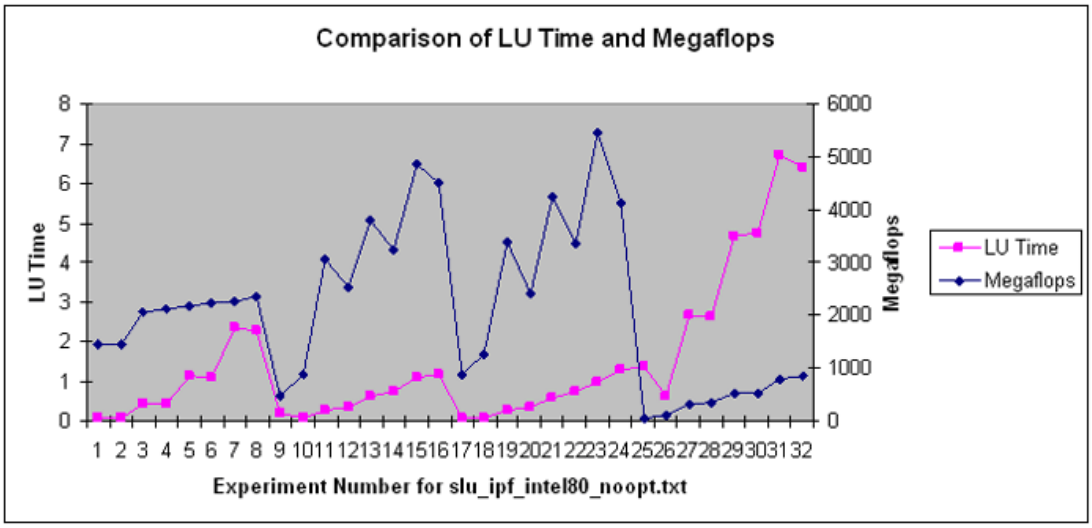

Figure 17 - Display of ScaLAPACK DATA from the Executable `xslu_ipf_intel80_noopt`

9.1 Gathering Instrumentation Data and Analyzing the ScaLAPACK Examples with the Intel® Trace Collector and Intel® Trace Analyzer

In the chapter titled **Interoperability of Intel® MPI Library with the Intel® Trace Collector and Intel® Trace Analyzer** cursory explanations were provided in gathering trace data and opening various analyzer panels for view trace file content. Analysis of the ScaLAPACK examples with Intel® Trace Collector and Intel® Trace Analyzer can also be done easily. This subsection will dwell further on the instrumentation and analysis process. The discussion will focus on how to alter the command-line options for the ScaLAPACK make command so that performance data collection will be possible. Note however, that the user will want to have plenty of disk storage available for collecting trace information on all of the examples because there are approximately 68 ScaLAPACK executables. To instrument the ScaLAPACK examples on an Intel® Itanium® 2-based cluster, use the following make command:

```
gmake lib64 F=intel80 LIBdir=/opt/intel/mkl72cluster/lib/64 MPILIB="-
L${VT_ROOT}/lib -lVT -ldwarf -lelf -lvtunwind"
```

where the above shell command should appear on one line. Recall the instrumentation processes discussed in Figure 1 and Figure 2. The recommended amount of disk storage for collecting trace data on all of the ScaLAPACK test cases is about 5 gigabytes. For an executable such as `xzgblu_ipf_intel80_noopt` that has been instrumented with the Intel® Trace Collector, a trace file called `xzgblu_ipf_intel80_noopt.stf` will be generated. Recalling the protocol that was discussed in the chapter for using Intel® Trace Analyzer, the user can proceed to analyze the content of `xzgblu_ipf_intel80_noopt.stf` with the following shell command:

```
traceanalyzer xzgblu_ipf_intel80_noopt.stf &
```

This command for invoking the Intel® Trace Analyzer will cause the Main panel display (Figure 18) to be produced as described previously:

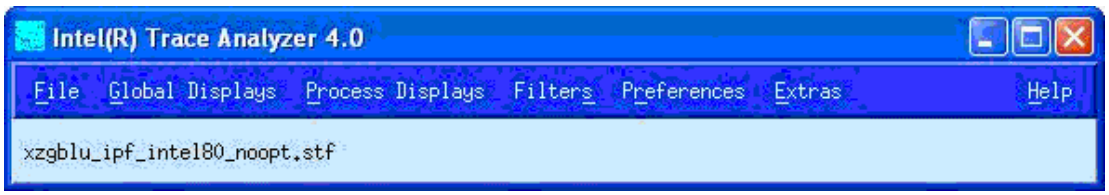

Figure 18 - Intel® Trace Analyzer Main Panel for the Executable `xzgblu_ipf_intel80_noopt`

along with a Frame Display (Figure 19). The user should again realize that the contents of a trace file such as `xzgblu_ipf_intel80_noopt.stf` will vary from cluster configuration to cluster configuration due to factors such as the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster.

Figure 19 – The Frame Display for the Executable `xzgblu_ipf_intel80_noopt`

Remember that a *frame* display should be considered a coarse view of the execution trace where each row in this panel represents a frame abstraction. The frames cover different sections of the STF trace data. The frames provide previews for these different parts and can be termed *thumbnails*. Usually several frames exist in one trace and the user will be able to navigate through the frames and select one or more to request additional detailed information. Notice that there is a simple color-coded legend provided in the upper right corner of the panel. The frame sections colored in green represent application activity and the red partitions depict MPI operations.

Clicking on the right-most mouse button will cause a context menu to appear. Within that context menu one can follow the selection path Load->Whole Trace as shown (Figure 20):

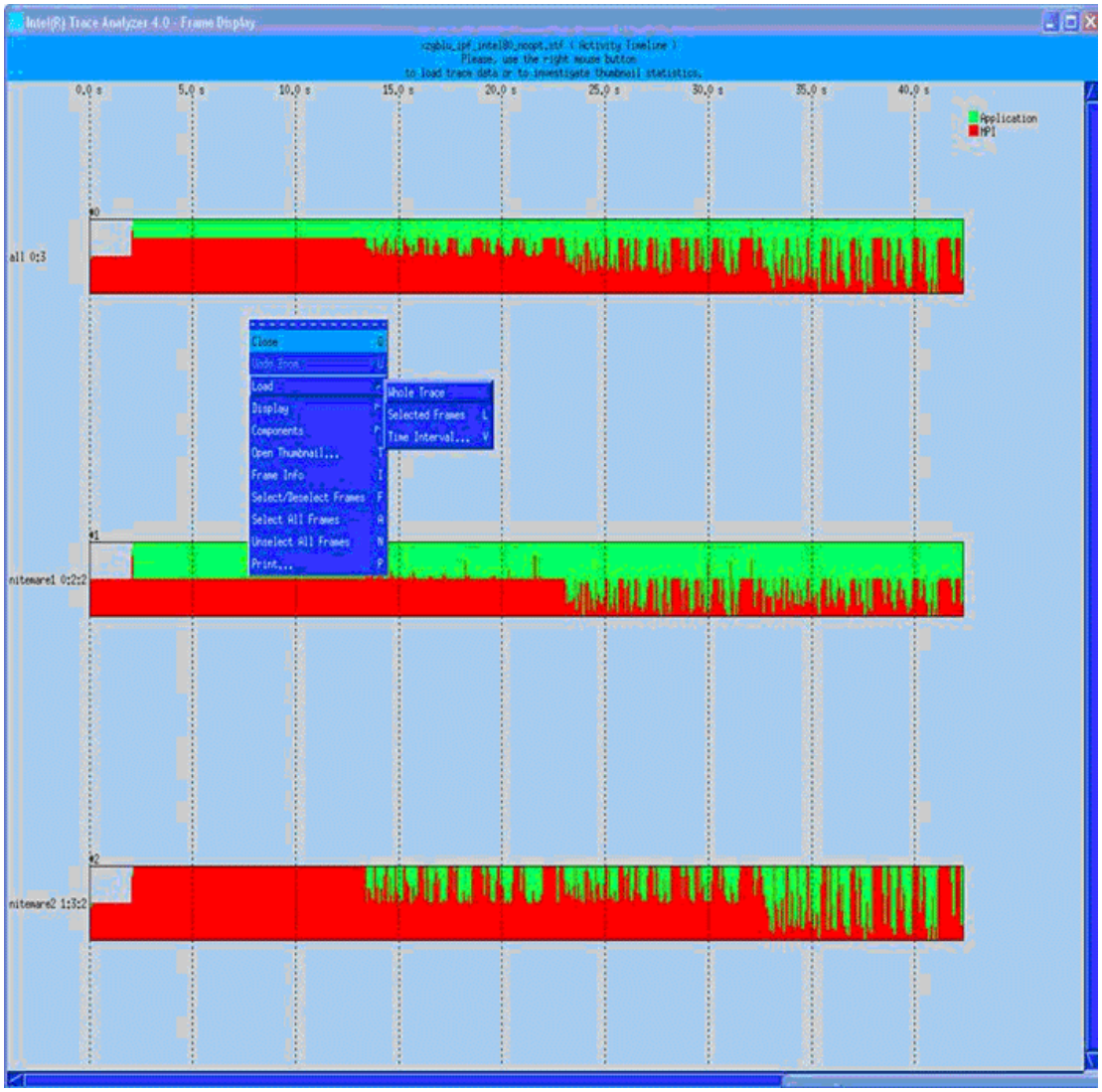

Figure 20 – Using the Context Menu to Load Data Frames for the Executable `xzgblu_ipf_intel80_noopt`

This will cause the frame information to be loaded. The result is a Summary Chart window (Figure 21):

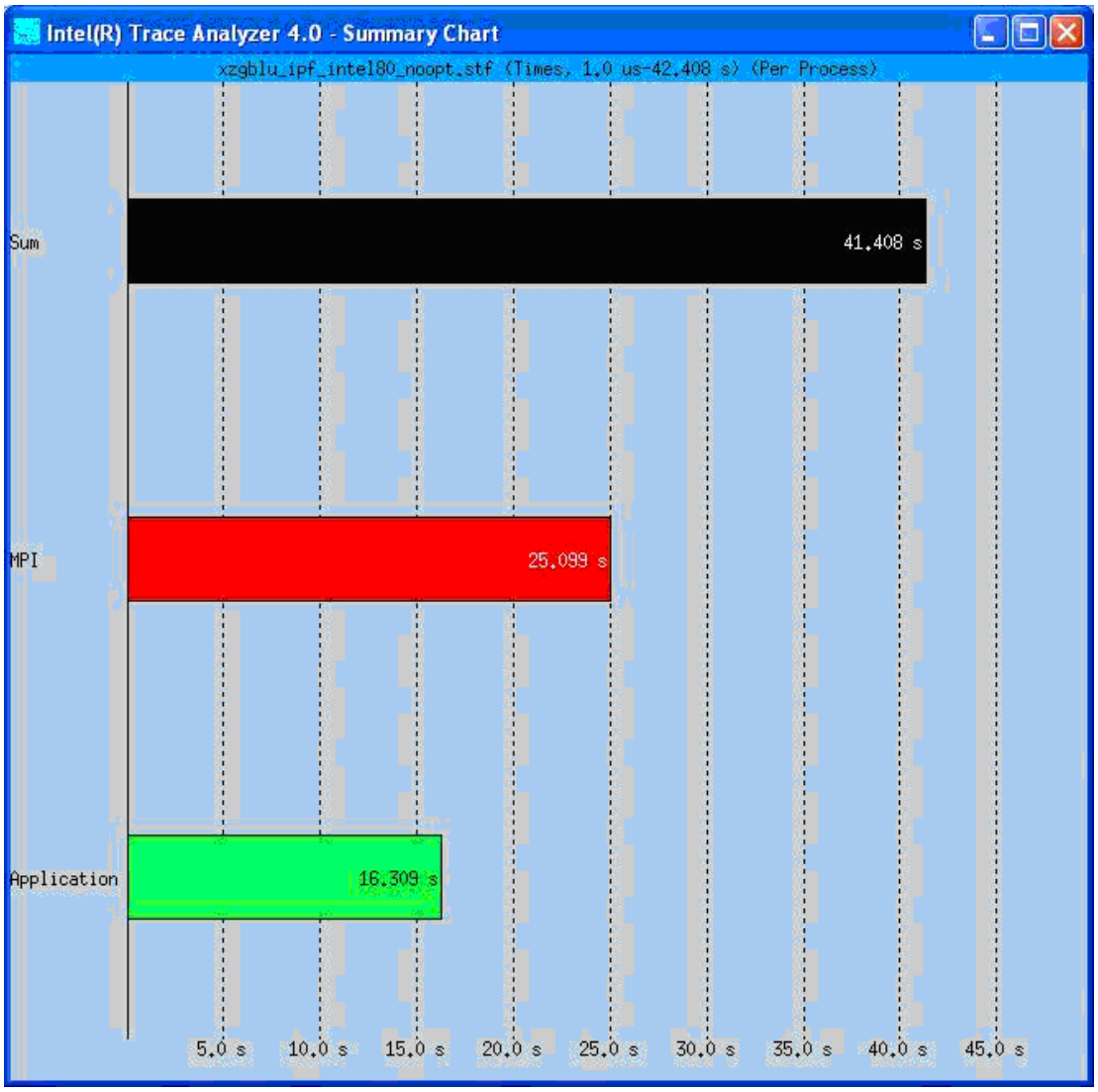

Figure 21 – The Summary Chart for the Executable `xzgblu_ipf_intel80_noopt`

This panel provides an overview of time durations that different components of the program executable used at run time.

For the Main panel display of the Intel® Trace Analyzer, one can proceed to select the Timeline view through the menu selection Global Displays->Timeline… or the user can press the hot key Ctrl-T. This will result in generating the Timeline display:

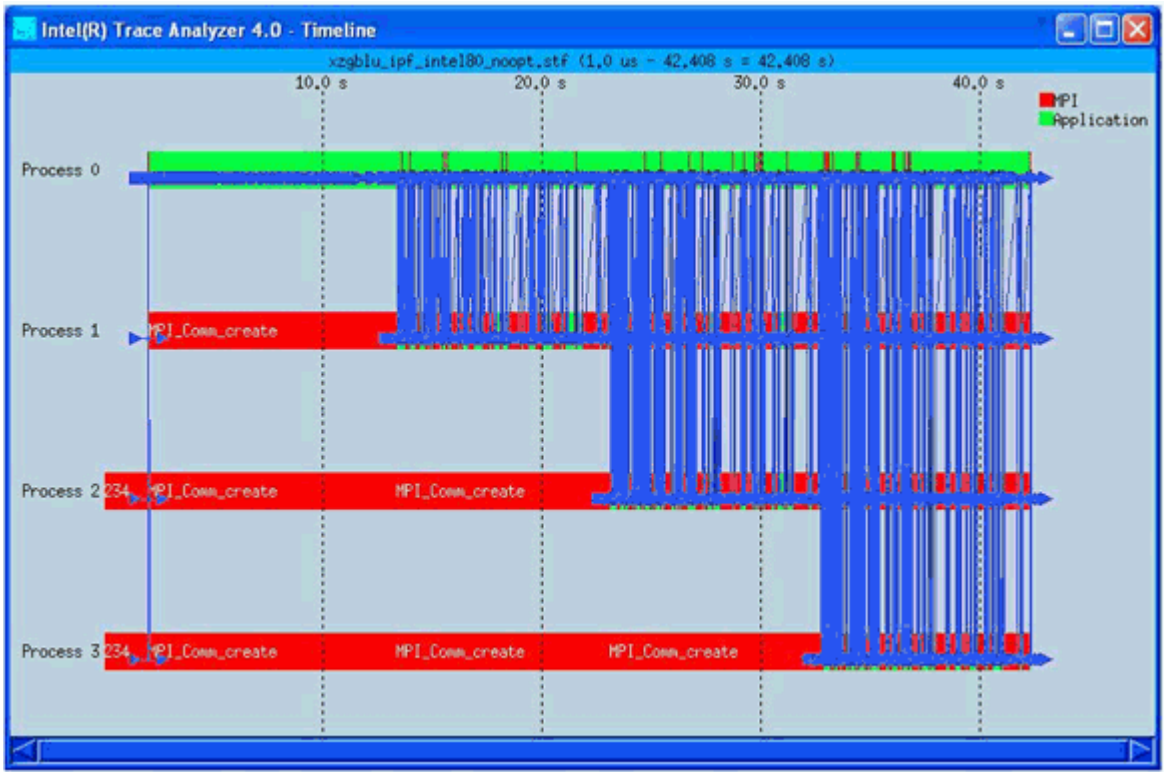

Figure 22 – The Timeline for the Executable `xzgblu_ipf_intel80_noopt`

In terms of review, the Timeline window (Figure 22) presents an overview of the entire execution trace. For each process, the display shows along the horizontal axis the different states and their change over execution time. Messages sent between processes are indicated as black lines connecting the sender and receiver processes. Global communications, such as broadcast or reduce operations, are grouped by purple lines and arrows. In both cases for local and global communications, the arrows indicate sender and receiver processes.

Since many messages are usually sent during execution of an application, the view of the timeline for the entire trace often looks cluttered as is illustrated in Figure 22. Since such a view is not particularly helpful, the Intel® Trace Analyzer provides the capability to limit the time portion that is displayed. This is essentially a zoom operation. The easiest way to use this mouse-zoom feature is to move the mouse pointer to the start of the interval you want to zoom into, press the left mouse button, and drag the mouse to the end of the zoom interval while continuing to press on the left mouse button. The Trace Analyzer will delimit the marked region that will be zoomed-in on with vertical bars. Finally, release the mouse button. The Timeline display will be redrawn showing just the time interval you selected and the contents will be magnified accordingly. For example, the time interval from 35.99 seconds to 36.187 seconds was selected in terms of a zoom operation (Figure 23). Recall that the black lines represent messages connecting the sender and receiver processes. These lines were masked by the purple global communication operations shown in Figure 22.

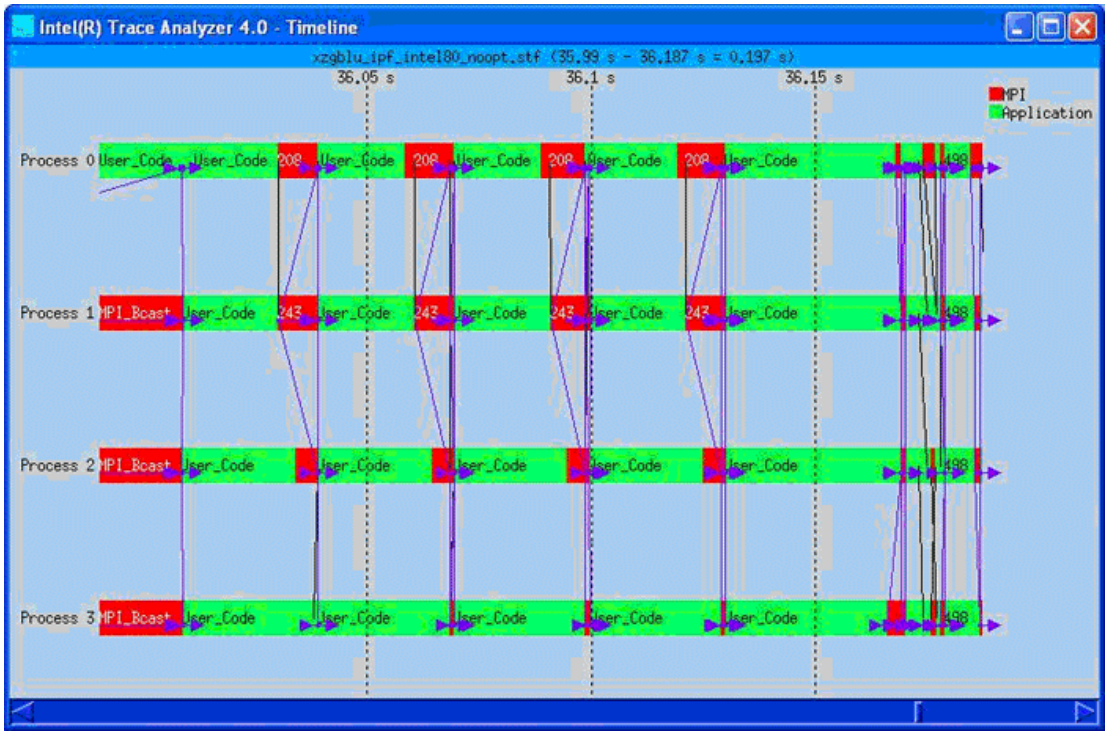

Figure 23 – Using the Zoom Operation on the Timeline Display for the Executable `xzgblu_ipf_intel80_noopt`

Figure 24, shows the Activity Chart Display. This panel is produced with the menu path Global Displays->Activity Chart or by using the hot key Ctrl-A. After doing this, the Pie Chart format is produced with the context menu (click on the rightmost button), and select Mode->Pie. In Figure 24, one sees a load imbalance between each of the MPI processes.

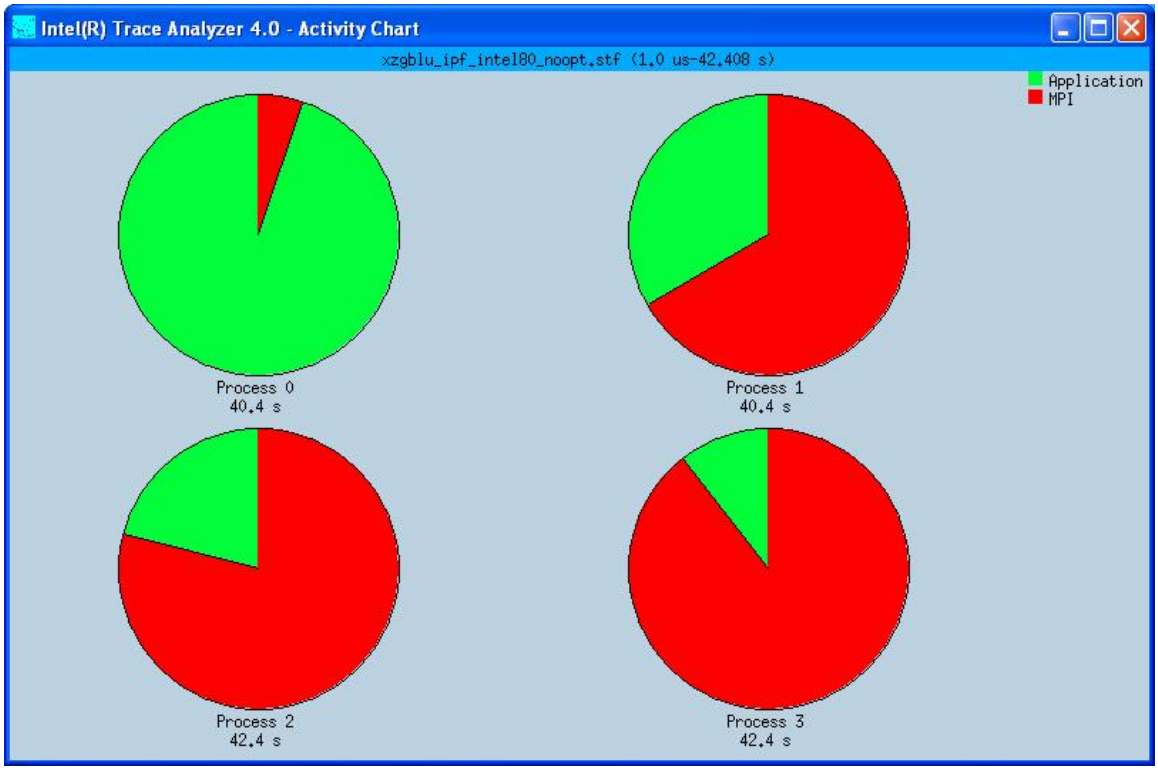

Figure 24 – The Activity Chart Display for the Executable `xzgblu_ipf_intel80_noopt`

To make inquiries about the Intel® Cluster Math Kernel Library 7.2, visit the URL: http://premier.intel.com.

10 Experimenting with the Intel® MPI Benchmarks

As mentioned in the general product capabilities section of the Intel Cluster Toolkit URL, the user should run Intel® MPI Library Benchmarks after installing the Intel® Cluster Tools to verify that the tools have been installed properly. Intel MPI Benchmarks will also tell the user how fast and efficient messaging is on their Intel® Pentium® 4, or Intel® Xeon™, or Intel® Itanium® 2, or Intel® EM64T cluster.

The Intel® MPI Benchmarks directory has 4 subfolders and a `ReadMe_first` file:

```
doc/ license/ ReadMe_first src/ versions_news/
```

If one changes their current working directory to that of src, one will find several files with the prefix of `make_...` where these files serve as include files to `Makefile`. If for example, one looks at the contents of `make_ia64`, one will see the default settings of the following makefile variables:

```
MPI_HOME     = ${MPICH}
MPI_INCLUDE  = $(MPI_HOME)/include
LIB_PATH     =
LIBS         =
CC           = ${MPI_HOME}/bin/mpicc
OPTFLAGS     = -O
CLINKER      = ${CC}
LDFLAGS      =
CPPFLAGS     =
```

At the top of the file called `Makefile`, one will see:

```
##### User configurable options #####
#include make_ia32
#include make_ia64
#include make_sun
#include make_solaris
#include make_dec
#include make_ibm_sp
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

For compilation on Itanium® 2 architectures for example, the user will need to remove the leading sharp symbol. Upon doing this, the beginning lines of `Makefile` should look something like the following:

```
##### User configurable options #####
#include make_ia32
include make_ia64
#include make_sun
#include make_solaris
#include make_dec
#include make_ibm_sp
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

That directory for the Intel® MPI Benchmarks should be titled IMB2.3. From that directory change to the child directory called `src`. In the child directory type the command:

```
make all MPI_HOME=/opt/intel_mpi_10 CC=mpiicc
```

The target called `all` will create the executables `IMB-MPI1`, `IMB-EXT`, and `IMB-IO`. Recall that `MPI_HOME` and `CC` are two of many makefile variables defined in the include file `make_ia64`. Thus, for the command line above, the default settings for `MPI_HOME` and `CC` are being overridden via the shell command-line. When compilation completes and the executables are built, one can then proceed to run the applications. For example, if the user wishes to run `IMB-MPI1` with 2 MPI processes, the following `mpiexec` command can be used:

```
mpiexec -n 2 ./IMB-MPI1 > IMB-MPI1.report 2>&1
```

The above command-line syntax assumes that a user is running Bourne Shell or Korn Shell. A partial list of the file `IMB-MPI1.report` might look something like:

```
#---------------------------------------------------
# Benchmarking PingPong
# ( #processes = 2 )
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000        45.06         0.00
            1         1000        45.06         0.02
            2         1000        45.06         0.04
            4         1000        47.10         0.08
            8         1000        45.06         0.17
           16         1000        47.10         0.32
           32         1000        49.15         0.62
           64         1000        51.20         1.19
          128         1000        57.34         2.13
          256         1000        67.58         3.61
          512         1000        88.06         5.54
         1024         1000       133.12         7.34
         2048         1000       221.18         8.83
         4096         1000       395.26         9.88
         8192         1000       745.47        10.48
        16384         1000      1445.89        10.81
        32768         1000      2840.58        11.00
        65536          640      5632.00        11.10
       131072          320     11296.00        11.07
       262144          160     22451.20        11.14
       524288           80     44774.40        11.17
      1048576           40     89497.60        11.17
      2097152           20    179814.40        11.12
      4194304           10    361267.20        11.07
```

The table above can be loaded into Microsoft Excel and the first and third columns can be used to build a two-dimensional Cartesian plot (Figure 25).

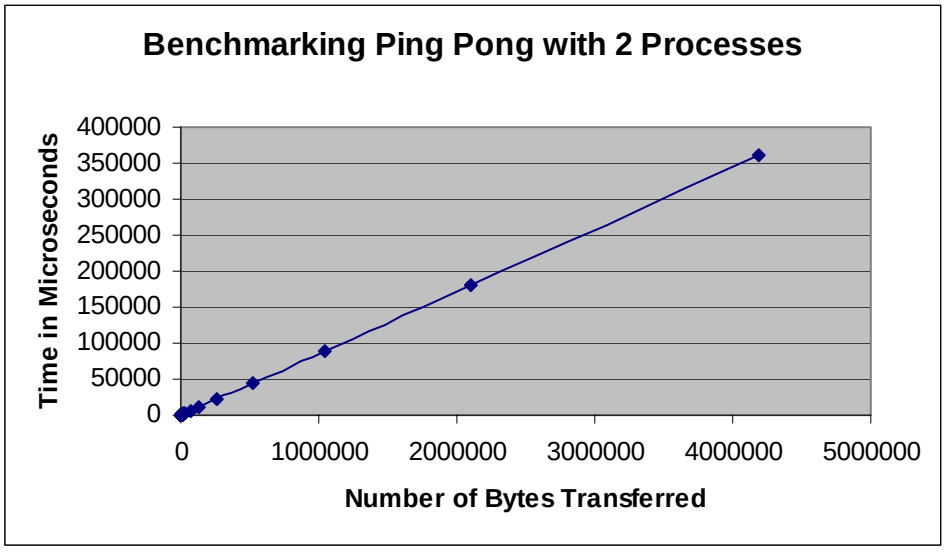

Figure 25 - Intel® MPI Benchmark Display for the Ping Pong Application Using 2 Processes

Please note that the above results are a function of the processor type, the memory configuration, competing processes, and the type of interconnection network between the nodes of the cluster.

The user can also run the executables for `IMB-EXT` and `IMB-IO` in a similar fashion:

```
mpiexec -n 2 ./IMB-EXT > IMB-EXT.report 2>&1
mpiexec -n 2 ./IMB-IO > IMB-IO.report 2>&1
```

An example segment of output for `IMB-EXT.report` might look something like:

```
#----------------------------------------------------------------
# Benchmarking Accumulate
# #processes = 2
#----------------------------------------------------------------
#
#    MODE: NON-AGGREGATE
#
       #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
            0          100        29.80       144.39        87.09
            4          100        59.02        59.23        59.13
            8          100        59.12        59.47        59.29
           16          100        59.30        59.56        59.43
           32          100        58.68        58.87        58.77
           64          100        59.74        60.02        59.88
          128          100        62.20        62.36        62.28
          256          100        63.41        63.72        63.56
          512          100        68.27        68.47        68.37
         1024          100        74.30        74.44        74.37
         2048          100        86.84        87.16        87.00
         4096          100       128.71       129.40       129.05
         8192          100       200.79       202.23       201.51
        16384          100       368.44       370.98       369.71
        32768          100       686.50       692.00       689.25
        65536          100      1342.41      1353.36      1347.89
       131072          100      2673.93      2685.88      2679.90
       262144          100      5230.01      5244.88      5237.45
       524288           80     10380.06     10397.12     10388.59
      1048576           40     22111.35     22170.87     22141.11
      2097152           20     45331.50     45614.80     45473.15
      4194304           10     95656.39     96885.20     96270.80
```

The partial contents of `IMB-IO.report` might look something like:

```
#---------------------------------------------------------------------------
# Benchmarking P_Write_Shared
# #processes = 2
#---------------------------------------------------------------------------
#
#    MODE: AGGREGATE
#
       #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]   Mbytes/sec
            0           50         0.24        16.20         8.22         0.00
            1           50         0.12       718.06       359.09         0.00
            2           50         0.14       582.06       291.10         0.00
            4           50         0.14       590.94       295.54         0.01
            8           50       699.22     24763.88     12731.55         0.00
           16           50        34.58     26002.86     13018.72         0.00
           32           50       427.04     24011.30     12219.17         0.00
           64           50      1071.58     26296.14     13683.86         0.00
          128           50        34.18    626092.42    313063.30         0.00
          256           50        38.78     26489.60     13264.19         0.01
          512           50        38.84     25962.62     13000.73         0.02
         1024           50        37.96    626598.54    313318.25         0.00
         2048           50       379.72     23274.08     11826.90         0.08
         4096           50        46.96    623217.28    311632.12         0.01
         8192           50        57.52    624184.58    312121.05         0.01
        16384           50        81.50     26217.00     13149.25         0.60
        32768           50      1106.82     34909.30     18008.06         0.90
        65536           50       719.18     57600.74     29159.96         1.09
       131072           50      1917.64    701683.64    351800.64         0.18
       262144           50      4234.72    196962.34    100598.53         1.27
       524288           32      8759.37    383840.94    196300.16         1.30
      1048576           16     16860.44    755549.76    386205.10         1.32
      2097152            8     33461.63   1481227.37    757344.50         1.35
      4194304            4     66036.22   2968876.24   1517456.23         1.35
      8388608            2    124223.59   5898820.52   3011522.05         1.36
```

Assuming that the user has an `rdma` device fabric installed on the cluster, the user can issue the following commands for the three Intel® MPI Benchmark executables so as to access that device fabric:

```
mpiexec -n 2 ./IMB-MPI1 -env I_MPI_DEVICE rdma > IMB-MPI1.rdma.report 2>&1
mpiexec -n 2 ./IMB-EXT -env I_MPI_DEVICE rdma > IMB-EXT.rdma.report 2>&1
mpiexec -n 2 ./IMB-IO -env I_MPI_DEVICE rdma > IMB-IO.rdma.report 2>&1
```

To instrument the Intel® MPI Benchmarks with the Intel® Trace Collector, the `make` command that was used previously will have to be extended. Recall that the basic link options needed to instrument an MPI application are:

```
-L${VT_ROOT}/lib -lVT -lvtunwind -ldwarf -lelf -lm –lpthread
```

This options need to be assimilated into the Intel MPI® Benchmarks makefile variables. This can be done as follows:

```
make clean
```

```
make all MPI_HOME=/opt/intel_mpi_10 CC=mpiicc OPTFLAGS="-O -g"
LIB_PATH="-L ${VT_ROOT}/lib" LIBS="-lVT -ldwarf -lelf -lvtunwind -
lpthread -lm"
```

Note that the second `make` command above is a single-line shell command. Assuming use of the Bourne Shell, the following environment variable settings, and commands will be used to gather instrumentation data.

```
export VT_LOGFILE_PREFIX=${PWD}/inst
export VT_PCTRACE=5
rm -rf ${VT_LOGFILE_PREFIX}
mkdir ${VT_LOGFILE_PREFIX}
mpiexec -n 2 ./IMB-MPI1 1> IMB-MPI1.inst.report 2>&1
mpiexec -n 2 ./IMB-EXT 1> IMB-EXT.inst.report 2>&1
```

As a reminder, the `VT_LOGFILE_PREFIX` environment variable will direct the instrumentation data to a sub-directory called `inst`. Also, the `VT_PCTRACE` environment variable enables runtime program counter tracing. The value of 5 indicates the number of call levels of procedures that will have source location information recorded. Since the unwinding of the call stack each time a function is called can be very costly the setting of the `VT_PCTRACE` environment variable should be handled with discretion. Additional documentation about Intel® Trace Collector environment variables can be found in the Intel® Trace Collector folder path:

<directory-path-to-ITC>/doc/Intel_Trace_Collector_Users_Guide.pdf

If one proceeds with the process of issuing the command:

```
traceanalyzer inst/IMB-MPI1.stf
```

and one proceeds to use the context menu to load the trace file from the Frame Display and generate a Timeline Display, one might see something like the following (Figure 26):

Figure 26 – Timeline display for the `IMB-MPI1` executable

One can zoom in by clicking on the index button of the mouse while simultaneously panning across the timeline (Figure 27). Figure 27 was produced by doing this panning operating on the left portion of Figure 26. The black lines signify message events that were sent from one process to another over the course of time. If one positions the cross-hairs of one of these black lines in Figure 27, and left-clicks on the mouse, one will see a pop-up window panel indicating that the message has been identified.

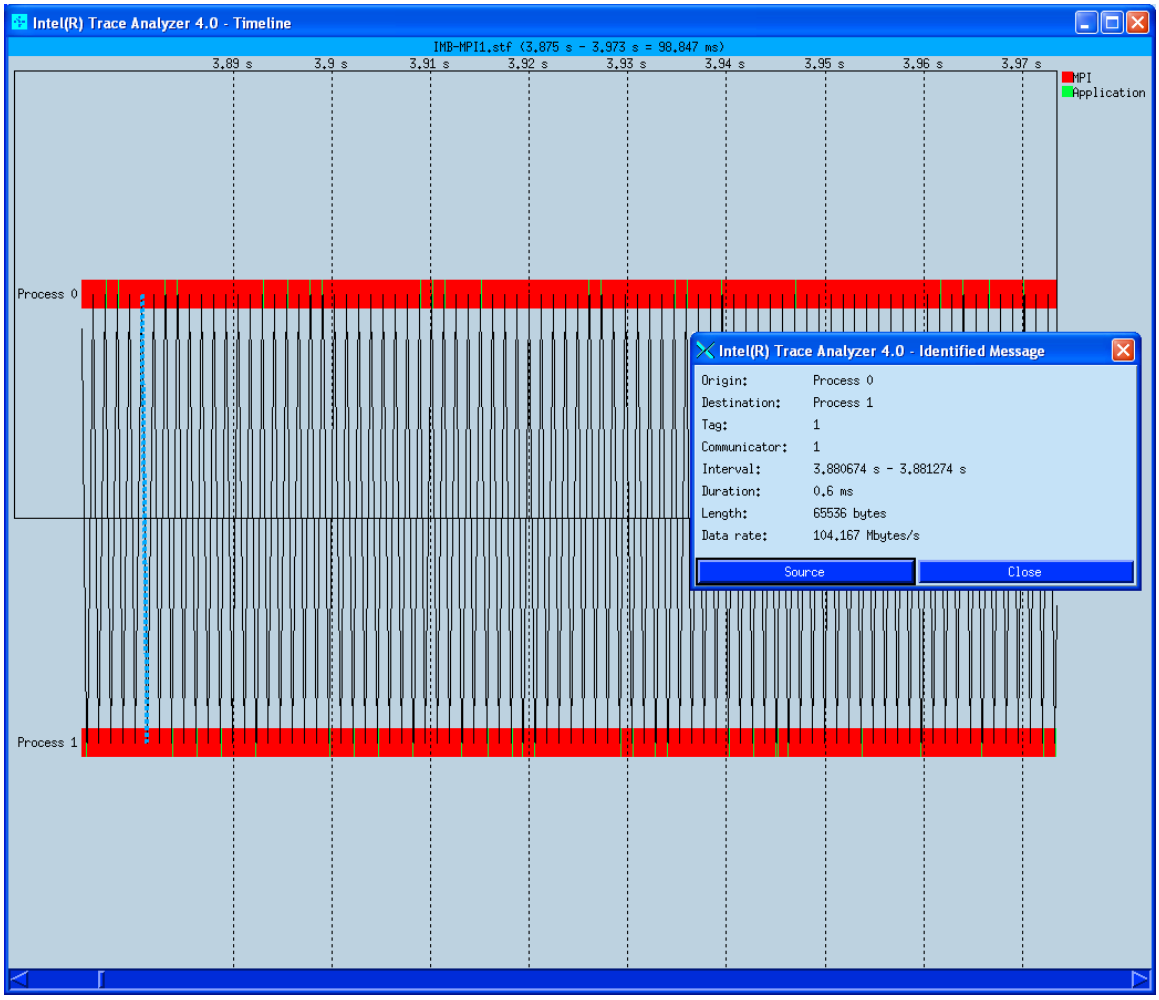

Figure 27 – Generation of Identified Message Window Panel for the Timeline Display

Figure 28, shows a close-up of the Identified Message panel that is displayed in Figure 27.

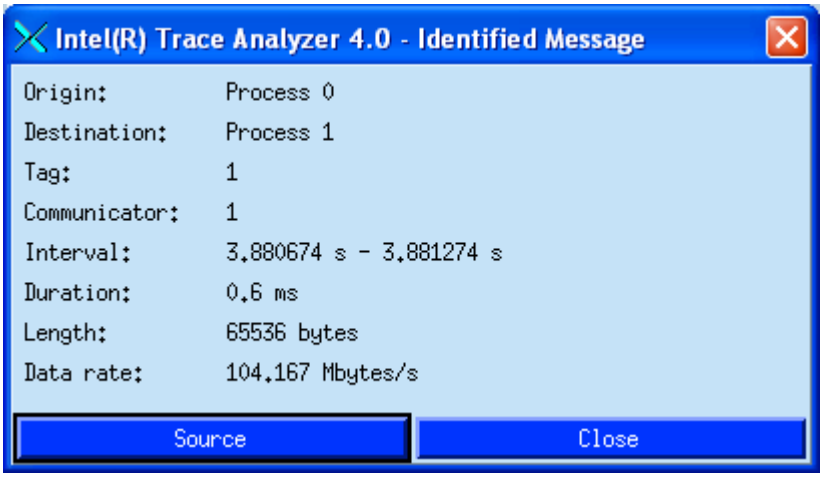

Figure 28 – Close-up view of the Identified Message panel

If one clicks on the Source button in the Identified Message Window in Figure 28, two Source View panels will be generated. This is illustrated in Figure 29 and Figure 30.

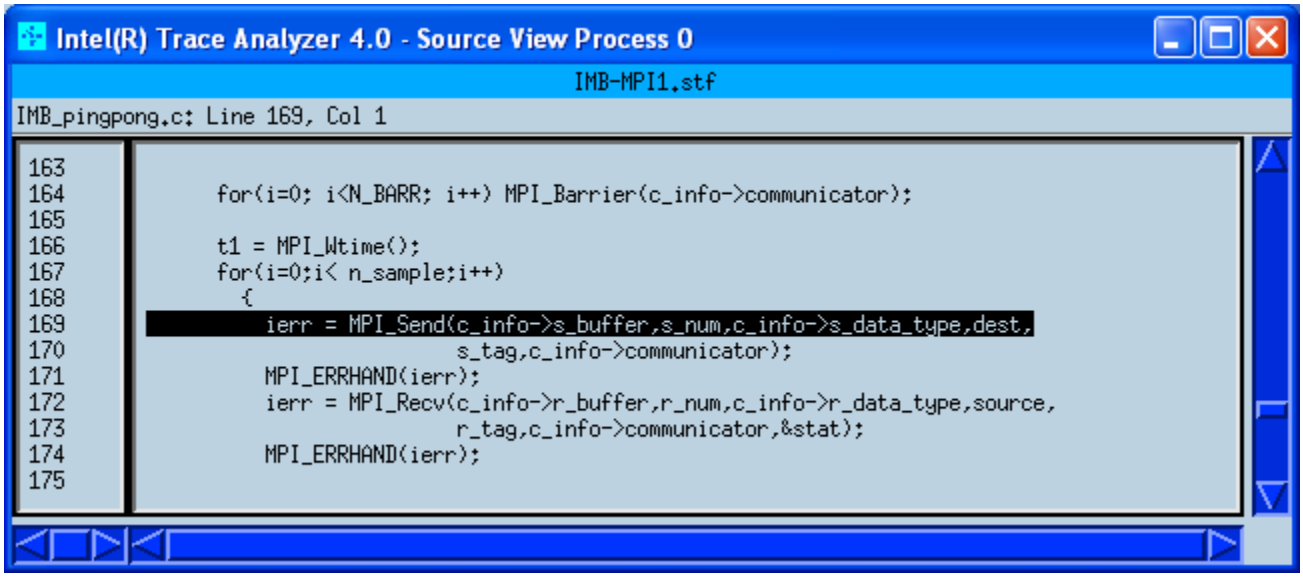

Figure 29 – Source View of code fragment that is sending a message

The Source View displays can be thought of as a drill down process where one goes from a graphical display in Figure 27 of black message lines to the actual source code fragments that are causing the message events.

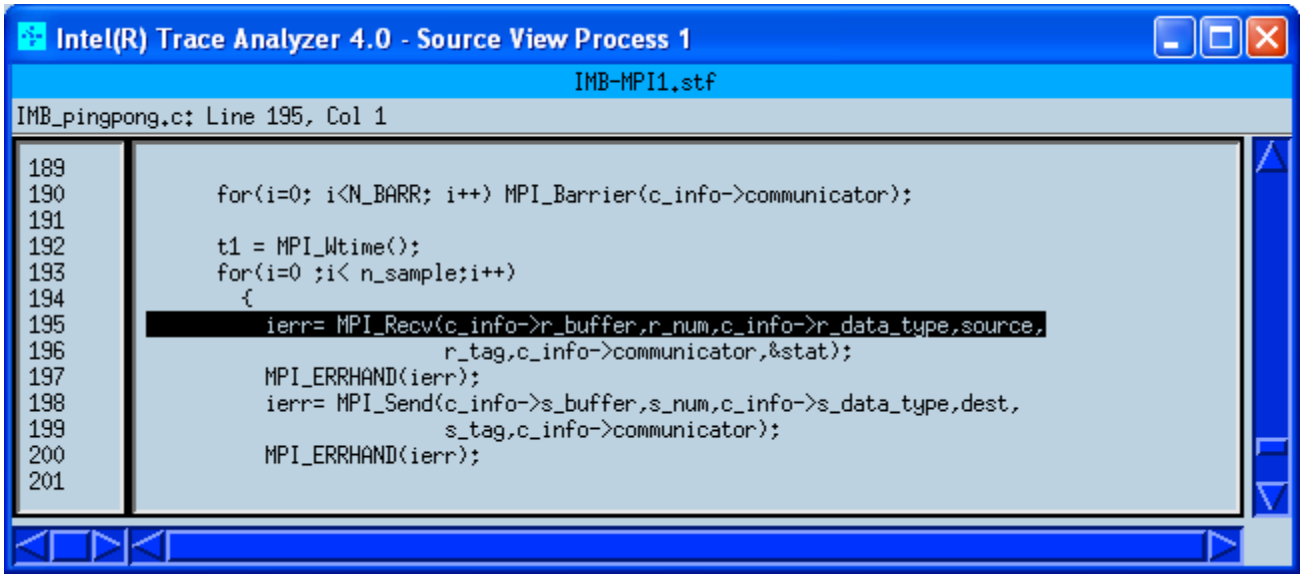

Figure 30 – Source View of code fragment that is receiving a message

To make inquiries about the Intel® MPI Benchmarks, visit the URL: http://premier.intel.com.

11 Hardware and Recommendations for Installation

Processor System Requirements

Intel® Pentium® 4 processor, or
Intel® Xeon™ processor, or
Intel® Itanium® 2 processor, or
Intel® EM64T processor

Note that it is assumed that the processors listed above are configured into homogeneous clusters.

Disk Space Requirements

10 GBs of disk space (minimum)

Operating System Requirements

Red Hat Enterprise Linux* 3.0 for Intel® Pentium® 4, or
Red Hat Enterprise Linux* 3.0 for Intel® Xeon™, or
Red Hat Enterprise Linux* 3.0 for Intel® Itanium® 2, or
Red Hat Enterprise Linux* 3.0 for Intel® EM64T (64 bit mode)

Memory Requirements

1 GB of RAM (minimum)

Intel® Compilers

For all of the Intel® processor architectures running Red Hat Enterprise Linux 3.0, the version number on the Intel® compilers should be 8.1 or greater.

12 System Administrator Checklist

1. Intel license keys should be place in a common repository for access by the software components of the Intel® Cluster Toolkit. An example license directory path might be:

   `/opt/intel/licenses`

13 User Checklist

1. Configure the environment variables. For the `~/.bashrc` file, an example of setting environment variables might be:

   ```
   export INTEL_LICENSE_FILE=/opt/intel/licenses
   . /opt/intel/ict/1.0/ictvars.sh
   ```

   Alternatively, for `~/.cshrc` one needs to set:

   ```
   setenv INTEL_LICENSE_FILE /opt/intel/licenses
   source /opt/intel/ict/1.0/ictvars.csh
   ```

   The above example-checklist items *assume* that `/opt/intel` is the default installation area.

Copyright © 2005 Intel Corporation. All rights reserved. Celeron, Chips, Dialogic, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel Centrino logo, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, Sound Mark, The Computer Inside, The Journey Inside, VTune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.